UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE LOGÍSTICA PARA LA PANADERÍA "DOS ESTRELLAS" APLICANDO TECNOLOGIAS DE INFORMACIÓN.

TESIS PARA OPTAR EL TÍTULO DE

INGENIERO INFORMÁTICO

PRESENTADO POR:

Br. MAURICIO APOLO BLANCA CAROLINA

PIURA – PERU

2016

UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE LOGÍSTICA PARA LA PANADERÍA "DOS ESTRELLAS" APLICANDO TECNOLOGIAS DE INFORMACIÓN.

TESIS PARA OPTAR EL TÍTULO DE

INGENIERO INFORMÁTICO

Blanca Carolina Mauricio Apolo
TESISTA

Ing. Jorge Luis Sandoval Rivera, Msc.
ASESORA

UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE LOGÍSTICA PARA LA PANADERÍA "DOS ESTRELLAS" APLICANDO TECNOLOGIAS DE INFORMACIÓN.

TESIS PARA OPTAR EL TÍTULO DE

INGENIERO INFORMÁTICO

Ing. Francisco Javier Cruz Vílchez, Msc.
PRESIDENTE

Ing. Esther Yolanda Lizana Puelles, Msc.
VOCAL

Ing. Nestor Manuel Castillo Burgos, Msc.
SECRETARIO

UNIVERSIDAD NACIONAL DE PIURA
FACULTAD DE INGENIERÍA INDUSTRIAL
DECANATO

ACTA DE SUSTENTACIÓN DE TESIS

Los Miembros del Jurado Calificador Ad-Hoc de la Tesis denominada: **«ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE LOGÍSTICA PARA LA PANADERÍA "DOS ESTRELLAS" APLICANDO TECNOLOGÍAS DE INFORMACIÓN»,** presentada por la señorita **BLANCA CAROLINA MAURICIO APOLO,** Bachiller de la Escuela Profesional en Ingeniería Informática; asesorada por el **Ing. Jorge Luis Sandoval Rivera, MSc.;** reunidos para la sustentación de ésta y luego de escuchar su exposición y las respuestas a las preguntas formuladas, la declaran:

UNIVERSIDAD NACIONAL DE PIURA
DECANATO
FACULTAD DE ING. INDUSTRIAL

Con el Calificativo: APROBADO

BUENO

En consecuencia la sustentante se encuentra apta para recibir el título profesional de **INGENIERO INFORMÁTICO**, conforme a Ley.

Piura, 08 de agosto del 2015

Ing. FRANCISCO JAVIER CRUZ VÍLCHEZ, MSc.
PRESIDENTE – JURADO CALIFICADOR

Ing. ESTHER YOLANDA LIZANA PUELLES, MSc.
VOCAL – JURADO CALIFICADOR

Ing. NÉSTOR MANUEL CASTILLO BURGOS, MSc.
SECRETARIO – JURADO CALIFICADOR

Contenido

DEDICATORIA

A mis queridos Padres.

Al apoyo incondicional de los seres más maravillosos que Dios puso sobre esta tierra, y que me acompañan día a día.

INTRODUCCION

En los últimos años se ha producido un rápido avance tecnológico, lo cual ha propiciado en las organizaciones que están acorde a este avance una mejora en la toma de decisiones teniendo resultados óptimos para la productividad de la organización y su pronto crecimiento.

No obstante en el sector de las micro y pequeñas empresas, muchas organizaciones se encuentran rezagadas tecnológicamente, dificultando su gestión y la posibilidad de competir exitosamente en el mercado.

La Panadería "Dos estrellas" es una de microempresa encargada de la producción y distribución de pan y derivados. Inaugurada desde el año 2010 siempre teniendo como objetivo principal ofrecer un producto de calidad y una atención de primera,

El presente trabajo llevara a los estudiantes a tener un conocimiento general de algunos aspectos importantes sobre el tema del proceso de gestión de logística de la panadería DOS ESTRELLAS.

La investigación planteada en esta tesis está relacionado a mejorar gestión de logística de la panadería DOS ESTRELLAS de la ciudad de Piura, a través de un sistema informático, para lo cual se tuvo que conocer en detalle el proceso en cuestión para llegar a esta desarrollarse e implementarse.

En el presente informe de Tesis se planteó el desarrollo del sistema de información para el proceso antes mencionado de la Panadería Dos Estrellas el cual se desarrolló con la metodología Tradicional RUP y codificado en Software libre como un aporte a la formalidad de las microempresas con el objetivo de controlar el stock de sus productos, mejorando tiempos y la satisfacción del personal de la empresa y sus clientes logrando un posicionamiento competitivo en el ámbito regional y satisfacer las necesidades de sus clientes.

CAPÍTULO 1.- PROBLEMA DE INVESTIGACION

1.1 PLANTEAMIENTO DEL PROBLEMA

Actualmente en la panadería "Dos Estrellas" se atienden pedidos a una gran cantidad de clientes finales, así como pequeños distribuidores de pan diariamente.
Para cumplir con estos pedidos de los clientes, se requieren de recursos tales como la materia prima que son adquiridos a distintos proveedores del rubro así como productos complementarios. Los proveedores se tienen registrados en cuadernos, además de sus productos, precios, direcciones, etc.
Existe mucha dificultad al momento de realizar una solicitud de materia prima ya que no se tiene a la mano estos registros, no hay un orden diario. Asimismo se presentan problemas al momento de realizar los pedidos por no tener un control adecuado de la cantidad de recurso que se necesita de acuerdo al número de clientes de todo tipo demandantes del producto.
En la atención de clientes principalmente a pequeños distribuidores no se lleva un control de cantidades solicitadas, que generalmente son realizadas a crédito, no se hace un seguimiento de los cobros a realizar por dichas ventas y las fechas de pedidos por atender. Esto dificulta una atención de calidad por parte de la empresa.

1.2 FORMULACION DEL PROBLEMA

¿Cómo mejorar el proceso de logística en la panadería "Dos Estrellas" haciendo uso de tecnologías de información?

1.3 OBJETIVOS DE LA APLICACIÓN

1.3.1. OBJETIVO GENERAL

Desarrollar un sistema de información para optimizar el proceso de logística de la panadería "Dos Estrellas" aplicando tecnologías de información.

1.3.2. OBJETIVOS ESPECÍFICOS

- Analizar el proceso de logística actual y determinar los requerimientos.
- Realizar el análisis y diseño del sistema así como de la Base de Datos.
- Desarrollar el sistema de información.
- Realizar pruebas.

1.4 OBJETIVOS DEL PRODUCTO DE LA INVESTIGACIÓN

- Facilitar en la panadería la eficaz adquisición de materia prima para la elaboración del pan en cantidades que satisfacen a todos los clientes.
- Tener en forma oportuna la información para la toma de decisiones al llevar un control adecuado de materia prima, proveedores, clientes, cobros y pagos a realizar.

1.5 HIPOTESIS

Hi: La gestión de logística de la panadería "Dos estrellas" si mejorará la toma de decisiones con la implementación un sistema de logística aplicando tecnologías de información.

Ho: La gestión de logística de la panadería "Dos estrellas" no mejorará la toma de decisiones con la implementación un sistema de logística aplicando tecnologías de información.

IDENTIFICACIÓN Y OPERACIONALIZACIÓN DE VARIABLE

- **Operacionalización de variables**

Variable (i) : Administración de logística	
Definición conceptual:	Se refiere a las actividades de control del movimiento de materiales y productos, desde el origen hasta su utilización por el cliente final.
Definición Operacional:	La medida de la Administración de logística se medirá en unidades de tiempo (Tiempo de procesos)
Indicador:	Eficiencia (Tiempo de procesos)

Tabla 1.5.1: Variable Administración Logística

Variable (d): Calidad del proceso de logística	
Definición conceptual:	Comprende en llevar un control a los usuarios para lograr tomar decisiones oportunas necesarias.
Definición Operacional:	Se medirá en el nivel de satisfacción de los usuarios.
Indicador:	Nivel de Satisfacción

Tabla 1.5.2: Variable Calidad de proceso de Logística

- **Indicadores de la Variable: Administración de logística**

Indicador: Tiempo de proceso	
Concepto:	Proporciona el tiempo en el cual se ejecutará el proceso de control de logística a los usuarios.
Obtención:	Cantidad de tiempo del proceso de abastecimiento actual Cantidad de tiempo del proceso de abastecimiento con SI

Tabla 1.5.3. Indicadores de Variable 1

- **Indicadores de la Variable: Calidad del proceso de logística**

Indicador: Nivel de Satisfacción	
Concepto:	Proporciona el nivel de satisfacción de los usuarios finales con la implementación del sistema de información.
Obtención:	Resultado que se obtiene con las entrevistas a la administración de la panadería "Dos Estrellas"

Tabla 1.5.4. Indicadores de Variable 2

CAPÍTULO 2.- MARCO TEÓRICO

2.1. MARCO REFERENCIAL

2.1.1. Panadería DOS ESTRELLAS

- RUC: 20529746238
- Representante Legal: CANTARO ARROYO DINO

La panadería DOS ESTRELLAS se fundó en el año 2010 por el Sr. Dino Cantaro Arroyo, teniendo al principio un pequeño local M Za. P Lote. 1-B Urb. Ignacio Merino II Etapa y que con trabajo arduo y constante actualmente cuenta con dos sucursales más en la Ciudad de Piura.

- **Misión de la empresa:**

Consolidarnos como la mejor panadería, ser reconocida por su calidad, con los más altos estándares de calidad higiene y servicio al cliente. Ser una empresa que se distinga en su género, colaborar en el crecimiento del país generando una pequeña fuente de empleo. Reafirmando el tema "Calidad Dos Estrellas"

- **Visión de la empresa:**

Ser la empresa panificadora líder en Piura, continuar, tener personal calificado, generar oportunidades de empleo. Aumentar el número de clientes que nos visitan en nuestras sucursales dando cada vez mejor servicio y la mayor atención.

2.1.2. Gestión de Stocks

Es necesario determinar otros aspectos relativos al momento y frecuencia con que debe efectuarse la compra. Estas cuestiones pueden ser efectivamente resueltas con una adecuada gestión de las existencias. Pero la finalidad principal de la gestión y control del inventario es determinar el nivel de existencias adecuado para minimizar las roturas de stocks y poder atender en todo momento a la demanda. Las roturas de stocks ocasionan pérdidas de beneficios, por las ventas que dejan de realizarse, o por la reducción en los márgenes de beneficio, ya que el producto suministrado ha tenido que obtenerse de forma urgente y con un coste adicional.

Estos costes de oportunidad disminuyen evidentemente a medida que los niveles de stocks aumentan. Pero al aumentar las existencias se incrementan otros costes, como el de almacenamiento, los intereses de los capitales invertidos, las primas de seguros, las mermas, y los derivados de la obsolescencia de los productos. Se plantea, por tanto, una contraposición de dos costes: los de oportunidad, que disminuyen al aumentar las existencias, y los de

mantenimiento del inventario que, al contrario, aumentan al incrementarse los stocks.

El objetivo a conseguir, por tanto, es minimizar la suma de ambos costes y no de cada uno de ellos por separado.

2.1.3. Logística

En la empresa, el concepto de logística hace referencia a las actividades de dirección del flujo de materiales y productos, desde la fuente hasta su utilización por el usuario final.

De un modo más preciso, se puede definir la logística como el arte de dirigir el flujo de materiales y productos de la fuente al usuario. El sistema logístico incluye el flujo total de materiales, desde la adquisición de las materias primas al suministro de productos acabados a los usuarios finales y los flujos de información que ocasionan el control y registro del movimiento de materiales.

El término distribución física suele utilizarse como sinónimo de logística, aunque de modo más preciso aquélla es sólo la parte de la logística que hace referencia al movimiento externo de los productos, desde el vendedor al cliente o comprador: es decir, a la logística relacionada con las actividades comerciales o logística comercial.

Descripción de las actividades de logística:

Las actividades logísticas dentro de una empresa se centran en tres de procesos básicos:

- Proceso de aprovisionamiento, la gestión de materiales entre los puntos de adquisición y las plantas de procesamiento que se tengan.
- Proceso de producción, gestión de las operaciones de fabricación de las diferentes plantas.
- Proceso de distribución, gestión de materiales entre las plantas mencionadas y los puntos de consumo.

Las técnicas logísticas en el proceso de aprovisionamiento y el proceso de distribución son muy similares y lo que pretende la logística es integrarlas y darle un grado alto de flexibilidad y rapidez de respuesta a las demandas del mercado.

- Actividades logísticas:
- Actividades Logísticas
- Proceso de pedidos
- Gestión de inventarios
- Transporte
- Servicio al cliente
- Compras
- Almacenamiento
- Planificación de productos

- Tratamiento de mercaderías

2.1.4. Gestión de la información

Características de las actividades logísticas:

El procesamiento de pedidos es la actividad que origina el movimiento de los productos y la realización de los servicios solicitados y, como recalcaremos posteriormente, tiene una gran incidencia en el tiempo de ciclo del pedido.

La gestión de inventarios tiene como objetivo principal proporcionar la disponibilidad requerida de los productos que solicita la demanda.

La actividad del transporte resulta indispensable en cualquier empresa para poder trasladar los materiales o productos propios, así como los productos finales (distribución).

La definición del nivel de servicio al cliente establece el nivel y la calidad de respuesta que deben tener todas las actividades de la cadena logística.

La actividad de compras afecta al canal de aprovisionamiento; a través de de ella se seleccionan las fuentes, se determinan las cantidades que es necesario adquirir, el momento de efectuar las adquisiciones y la planificación de los productos. De acuerdo con el canal de distribución se establece la cuantía de los componentes y la secuencia y el ciclo de producción, lo cual repercute en el funcionamiento logístico global, pero en particular, en la gestión de inventarios y la eficacia del transporte; es por esto que a veces, las dos actividades son consideradas como funciones del departamento de producción.

El almacenamiento comporta las decisiones asociadas tales como la determinación del espacio requerido, el diseño y la configuración de los almacenes y la disposición de los productos en su interior. Es una actividad de los productos defectuosos.

La gestión de la información abarca la recorrida, el almacenamiento, el tratamiento y el análisis de los datos necesarios para desarrollar la planificación y el control, lo cual da soporte a todo el sistema logístico.

Flujos de materiales, productos e información en el sistema logístico:

Considerando simultáneamente los dos grandes objetivos de la logística, un nivel de servicio al cliente que maximice las ventas y minimice los costes, es conveniente diseñar, planificar y controlar una red de distribución que permita que, situando los productos en su destino en el momento preciso, se consigan ambos objetivos al máximo nivel.

Esta red constituye una determinada configuración de puntos de fabricación, almacenamiento y ventas, y un sistema de transporte y de tratamiento de la información adecuada, cuyo funcionamiento global ha de ser efectivo y eficiente para cumplir con los objetivos enunciados.

Existe una red genérica a base de centros (que representan proveedores, factorías, almacenes o puntos de ventas, en los cuales se detiene temporalmente el flujo de los productos) y los enlaces entre ellos, que indican el movimiento al que están sometidas las mercaderías. Pueden situarse

diversos enlaces entre parejas de centros para materializar la posibilidad de un tráfico de diferentes productos o la existencia de otras trayectorias o alternativas de transporte.

Es necesario decir que el flujo de materiales y productos se produce básicamente en el sentido del suministro a la demanda, y que se denomina descendente por el hecho de estar dirigido hacia el consumidor, situado en el canal.

Por otro lado, existe otra red muy parecida desde el punto de vista conceptual, la de información, por la cual y a través de enlaces transita la información relacionada con la gestión de las diferentes actividades logísticas que se intercambian entre todos los lugares de recepción y que son los centros de esta red.

El flujo de información, se dirige principalmente desde el consumidor hasta el lugar de origen de los suministros, por eso ahora se denomina ascendente. Los sistemas que le dan soporte constituyen una parte esencial de la organización, ya que proporcionan los elementos de juicio requeridos para los procesos de toma de decisiones, coordinación y control, y posibilitan con su gestión rápida y eficaz, la integración correcta de todos los centros de la actividad empresarial. El sistema logístico total resulta, de la combinación de las dos redes.

2.1.5. Desarrollo del proceso logístico:

Para desarrollar de la manera más adecuada el proceso logístico, primero se ha de considerar en su totalidad, evaluando los puntos requeridos para establecerlos, los medios que se pondrán en juego, el volumen de operaciones resultantes y sus diferentes fases, así como las previsiones futuras y el tráfico.

Se podrá hacer la distribución de los centros de producción y la ordenación relativa que han de mantener, para acabar efectuando la distribución en planta de los diferentes puntos y de las mercaderías y productos que son afectados.

Por otro lado, de las etapas de transporte y almacenamiento (o el escalonamiento que se haya fijado en el plan), dependerá el número y la localización más adecuada de los almacenes, las dimensiones, el diseño y la implantación que se ha de realizar. Se especificará a qué procesos concretos tendrán que responder, obviamente en función de los productos, la forma de presentación, etc., pero que en general pueden ser la entrada del material, su descarga y recepción, el control de las mercaderías y su embalaje para el almacenamiento.

También tendremos en cuenta los procesos de traslado de los elementos a la zona de distribución con el fin de preparar los envíos y el control de sus salidas, requeridos por las expediciones que se carguen en los medios de transporte utilizados.

2.2. MARCO CONCEPTUAL

2.2.1. Metodologías Tradicionales

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales.

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada

Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar.

Entre las metodologías tradicionales se encuentran:

- RUP (Rational Unified Procces)
- MSF (Microsoft Solution Framework)
- Win-Win Spiral Model
- Iconix

En el presente trabajo trabajaremos con la metodología RUP ya que es una de las formas más disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo.

2.2.2. Metodología RUP (RATIONAL UNIFIED PROCESS)

Es una metodología cuyo fin es entregar un producto de software. Se estructura todos los procesos y se mide la eficiencia de la organización.

Es un proceso de desarrollo de software el cual utiliza el lenguaje unificado de modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP es un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Describe cómo aplicar enfoques para el desarrollo del software, llevando a cabo unos pasos para su realización.

Se centra en la producción y mantenimiento de modelos del sistema.

La metodología RUP es dirigida por casos de uso, estos son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso

representan los requisitos funcionales del sistema.

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema.

También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo.

2.2.3. Lenguaje de modelamiento unificado UML

UML (Unified Modeling Language) es un lenguaje para especificar, visualizar, construir y documentar las diferentes etapas del desarrollo de software, así como para modelado de procesos de negocio u otros sistemas no-software. UML reúne una colección de las mejores prácticas en la ingeniería que han sido utilizadas con éxito para modelar sistemas grandes y complejos, ya que cubre tanto objetos conceptuales como los procesos de negocio y funciones del sistema, como también objetos concretos como clases en un lenguaje de programación, esquemas de base de datos y componentes reusables de software.

UML ha sido creado por los expertos en la metodología orientada a objetos, utilizando información de otros importantes expertos en metodología, vendedores de software, y usuarios finales. Su objetivo era unificar los diversos sistemas que había y crear un lenguaje de modelado con las mejores características de cada uno.

El UML fue adoptado por el OMG (Object Management Group) como estándar en noviembre de 1997 y ha comenzado rápidamente a ser utilizado en el diseño, especificación, construcción, visualización y documentación de software.

La técnica central en el UML es el Modelamiento en Objetos que es un lenguaje que permite la especificación de clases, sus datos o atributos (privados) y métodos (públicos), herencia y otras relaciones entre las clases.

UML se compone de muchos elementos de esquematización que representan las diferentes partes de un sistema de software. Los elementos UML se utilizan para crear diagramas, que representa alguna parte o punto de vista del sistema:

- Diagrama de casos de uso que muestra a los actores (otros usuarios del sistema), los casos de uso (las situaciones que se producen cuando utilizan el sistema) y sus relaciones.
- Diagrama de clases que muestra las clases y la relaciones entre ellas.
- Diagrama de secuencia muestra los objetos y sus múltiples relaciones entre ellos.
- Diagrama de colaboración que muestra objetos y sus relaciones, destacando los objetos que participan en el intercambio de mensajes.
- Diagrama de estado muestra estados, cambios de estado y eventos en un objeto o en parte del sistema.
- Diagrama de actividad que muestra actividades, así como los cambios de una a otra actividad junto con los eventos que ocurren en ciertas partes del sistema.

- Diagrama de componentes que muestra los componentes de mayor nivel de la programación (cosas como Kparts o Java Beans).
- Diagrama de implementación que muestra las instancias de los componentes y sus relaciones.
- Diagrama de relaciones de entidad que muestra los datos y las relaciones y restricciones entre ellos.

2.2.4. Lenguaje de programación JAVA:

Java es un lenguaje de programación creado para satisfacer una necesidad de la época planteada por nuevos requerimientos hacia los lenguajes existentes. Antes de la aparición de Java, existían otros importantes lenguajes (muchos se utilizan todavía). Entre ellos el lenguaje C era probablemente el más popular debido a su versatilidad; contiene posibilidades semejantes a programar en ensamblador, pero con las comodidades de los lenguajes de alto nivel. Uno de los principales problemas del lenguaje C (como el de otros muchos lenguajes) era que cuando la aplicación crecía, el código era muy difícil de manejar. Las técnicas de programación estructurada y programación modular, paliaban algo el problema. Pero fue la programación orientada a objetos la que mejoró notablemente la situación.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales, y librerías de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Estructuralmente, el lenguaje Java comienza con paquetes. Un paquete es el mecanismo de espacio de nombres del lenguaje Java. Dentro de los paquetes se encuentran las clases y dentro de las clases se encuentran métodos, variables, constantes, entre otros.

2.2.5. Java Archive

Un archivo JAR, o Java Archive en inglés, es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. Los archivos JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar. Existen tres operaciones básicas con este tipo de

archivos: ver contenido, comprimir y descomprimir.

La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados, es decir que el intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.

En muchos casos, los archivos JAR no son solo simples archivos de archivos y / o recursos de clases de Java. Se utilizan como bloques de construcción para las aplicaciones y extensiones. Muchas veces se utilizan para paquete de tienda y de configuración de extensión de datos, incluida la seguridad, control de versiones, extensión y servicios.

Jcalendar:

JCalendar es un Java bean selector de fechas para recoger gráficamente una fecha. JCalendar se compone de varios otros granos de Java, un JDayChooser, un JMonthChooser y JYearChooser. Todos estos granos tienen una propiedad locale, proporcionar varios iconos y su propio editor de propiedades de configuración regional. Por lo que fácilmente se pueden utilizar en los constructores GUI. También forma parte del paquete es un JDateChooser, un grano de compuesto de un IDateEditor (para la edición de fecha directa) y un botón para la apertura de una JCalendar para seleccionar la fecha.

Este programa es software libre; puedes redistribuirlo y / o modificarlo bajo los términos de la Licencia Pública General de GNU según es publicada por la Free Software Foundation.

ItextPDF

iText es una biblioteca Open Source para crear y manipular archivos PDF, RTF, y HTML en Java. Fue escrita por Bruno Lowagie, Paulo Soares, y otros; está distribuida bajo la Affero General Public License.

Ha sido extendida a una biblioteca PDF de propósito general, capaz de rellenar formularios, mover páginas de un PDF a otro, y otras cosas. Estas extensiones son a menudo mutuamente excluyentes. Una clase te permite rellenar en formularios, mientras una clase diferente e incompatible hace posible copiar páginas de un PDF a otro.

El soporte de PDF de iText es, sin embargo, bastante extensivo. Esto soporta firmas basadas en PKI de PDF, cifrado de 40-bit y 128-bit, corrección de colores, PDF/X, gestión de colores por perfiles ICC, y es anfitriona de otras características.

2.2.6. Programación orientada a objetos:

Orientado a objetos, se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos.

Un objeto puede verse como un paquete que contiene el "comportamiento" y el "estado". El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa.

Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software:

- La falta de portabilidad del código y su escasa reusabilidad.
- Código que es difícil de modificar.
- Ciclos de desarrollo largos.
- Técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de

sortear es usualmente la herencia. El concepto de programación orientada a objetos (POO) no es nuevo, lenguajes clásicos como SmallTalk se basan en ella. Dado que la POO se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo. No debemos confundir que esté basado en objetos con que sea orientado a objetos: para que sea orientado a objetos al margen que esté basado en objetos, necesita tener clases y relaciones de herencia entre ellas. Hemos utilizado con mucha frecuencia la palabra paradigma, que convendría formalizar de alguna forma: Se entiende paradigma como un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, esto es, una forma de ver el mundo. La visión OO nos obliga a reconsiderar nuestras ideas acerca de la computación, de lo que significa ponerla en práctica y de cómo debería estructurarse la información en los sistemas de información.

Los objetos

El elemento fundamental de la POO es, como su nombre indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización. En este caso las estructuras de datos y los algoritmos usados para manipularlas están encapsulados en una idea común llamada objeto. Esta definición especifica dos propiedades características de los objetos:

- En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados.
- En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

Un objeto está compuesto de:

- Atributos: como podrían ser las coordenadas que definen el centro del círculo y el radio.
- Métodos: por ejemplo, el que nos permite leer o modificar los valores de estos atributos, dibujarlo o bien calcular su circunferencia o su área.

Son conceptos fundamentales en este paradigma la idea de encapsulación, en cuanto a que atributos y métodos forman el objeto como una única unidad y el de ocultación de información en cuanto que los atributos deben estar lo más ocultos posibles al exterior y sólo ser manipulados o consultados a través de los métodos pertinentes que son los que conforman la interfaz del objeto.

2.2.7. Patrones de diseño

Cuando se realiza un modelo de objetos para un sistema informático se suelen utilizar objetos del mundo real, es decir, si en el caso hipotético que se desee un sistema para administrar la información de una biblioteca se basaría en estantes, libros y dependiendo de las necesidades se estarían incluyendo muchos otros objetos.

Si se tratara solo del objeto libro, el cual contiene la propiedad nombre. Con esto se tiene un modelo funcional para la administración de libros, naturalmente toca crear algunas interfaces gráficas y formularios para que un usuario pueda acceder a la aplicación y también guardar la información de los libros en algún lugar para que el usuario no pierda su información cuando salga del aplicativo o se apague el equipo en el que trabaja.

La solución a este problema es simple, utilizar un objeto para administrar la persistencia (DAO) y otro para manejar los datos como una unidad (DTO).

Objeto de Transferencia de Datos (DTO)

El Objeto de Transferencia de Datos (DTO) es un objeto que transporta datos entre procesos. La motivación de su uso tiene relación con el hecho que la comunicación entre procesos es usualmente realizada mediante interfaces remotas, donde cada llamada es una operación costosa. Como la mayor parte del costo de cada llamada está relacionado con el tiempo round-trip entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola invocación.

La diferencia entre DTO y Objetos de Negocio (DAO) es que un DTO no tiene más comportamiento que almacenar y entregar sus propios datos.

Los DTOs son objetos simples que no deben contener lógica de negocio que requiera pruebas generales.

Objeto de Acceso a Datos (DAO)

En software de computadores, un , Objeto de Acceso a Datos (DAO) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

La mayoría de las aplicaciones, tienen que persistir datos en algún momento, ya sea serializándolos, guardándolos en una base de datos relacional, o una base de datos orientada a objetos, etc. Para hacer esto, la aplicación interactúa con la base de datos. El "como interactúa" NO debe ser asunto de la capa de lógica de negocio de la aplicación, ya que para eso está la capa de persistencia, que es la encargada de interactuar con la base de datos. DAO es un patrón de diseño utilizado para crear esta capa de persistencia.

DAO encapsula el acceso a la base de datos. Por lo que cuando la capa de lógica de negocio

necesite interactuar con la base de datos, va a hacerlo a través de la API que le ofrece DAO. Generalmente esta API consiste en métodos CRUD (Create, Read, Update y Delete). Entonces por ejemplo cuando la capa de lógica de negocio necesite guardar un dato en la base de datos, va a llamar a un método create(). Lo que haga este método, es problema de DAO y depende de cómo DAO implemente el método create(), puede que lo implemente de manera que los datos se almacenen en una base de datos relacional como puede que lo implemente de manera que los datos se almacenen en ficheros de texto. Lo importante es que la capa de lógica de negocio no tiene por qué saberlo, lo único que sabe es que el método create() va a guardar los datos, así como el método delete() va a eliminarlos, el método update() actualizarlos, etc. Pero no tiene idea de cómo interactúa DAO con la base de datos.

En una aplicación, hay tantos DAOs como modelos. Es decir, en una base de datos relacional, por cada tabla, habría un DAO.

DAO consiste básicamente en una clase que es la que interactúa con la base de datos. Los métodos de esta clase dependen de la aplicación y de lo que queramos hacer. Pero generalmente se implementan los métodos CRUD para realizar las "4 operaciones básicas" de una base de datos.

Los DTO o también denominados VO (Value Object). Son utilizados por DAO para transportar los datos desde la base de datos hacia la capa de lógica de negocio y viceversa. Por ejemplo, cuando la capa de lógica de negocio llama al método create(), un DAO inserta un nuevo dato que la capa de lógica de negocio le pasa como parámetro a través de un DTO.

2.2.8. Plataforma:

Java es un lenguaje de programación con independencia de plataforma, que significa que programas escritos este lenguaje pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run anywhere".

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode) instrucciones máquina simplificada específica de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino, que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

La licencia sobre Java de Sun insiste que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la

implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ''dependientes'' de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios así como una orden judicial forzando la acatación de la licencia de Sun. Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector o plugin aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o "compilación al vuelo"), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una "recompilación dinámica" en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (en español: "Escríbelo una vez, ejecútalo en cualquier parte" por "Escríbelo una vez, depúralo en todas partes").

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

2.2.9. Entorno de desarrollo: NETBEANS

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans

IDE2 es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos (Actualmente Sun Microsystems es administrado por Oracle Corporation).

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

El NetBeans IDE es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

Framework esta simplificando el desarrollo de aplicaciones para escritorio Java Swing. El paquete de NetBeans IDE para Java SE contiene lo que se necesita para empezar a desarrollar plugins y aplicaciones basadas en la plataforma NetBeans; no se requiere un SDK adicional.

Las aplicaciones pueden instalar módulos dinámicamente. Algunas aplicaciones pueden incluir un módulo de actualización para permitir a los usuarios descargar Actualizaciones de firma digital y nuevas características directamente dentro de la aplicación en ejecución. Reinstalando una actualización o una nueva versión, no obligando a los usuarios a descargar toda la aplicación de nuevo.

La plataforma ofrece servicios reusables comunes para las aplicaciones de escritorio, permitiendo a los desarrolladores centrarse en la lógica de sus aplicaciones. Algunas de las características de la aplicación son:

- Gestión de la interfaz de usuario (menús y barras de herramientas)
- Gestión de configuración de usuario
- Gestión de almacenamiento (guardar o cargar algún tipo de dato)
- Gestión de ventana
- Marco Asistente (soporta diálogos para a paso)
- Librería visual de Netbeans
- Herramientas de desarrollo integrado

NetBeans IDE es libre, código abierto, multiplataforma con soporte integrado para el lenguaje de programación Java.

2.2.10. Encriptación

Encriptación es el proceso mediante el cual cierta información o texto sin formato es cifrado de forma que el resultado sea ilegible a menos que se conozcan los datos necesarios para su interpretación. Es una medida de seguridad utilizada para que al momento de almacenar o transmitir información sensible ésta no pueda ser obtenida con facilidad por terceros. Opcionalmente puede existir además un proceso de desencriptación a través del cual la información puede ser interpretada de nuevo a su estado original, aunque existen métodos de encriptación que no pueden ser revertidos. El término encriptación es traducción literal del inglés y no existe en el idioma español. La forma más correcta de utilizar este término sería cifrado.

Para encriptar datos, existen muchos métodos diferentes, para el sistema presentado se ha elegido usa el método de encriptación MD5 que será detallado a continuación:

MD5

En criptografía, MD5 que significa "Algoritmo de Resumen del Mensaje 5" es un algoritmo de reducción criptográfico de 128 bits ampliamente usado. Es uno de los algoritmos de reducción criptográficos diseñados por el profesor Ronald Rivest del MIT (Instituto Tecnológico de Massachusetts). Fue desarrollado en 1991 como reemplazo del algoritmo MD4 después de que Hans Dobbertin descubriese su debilidad.

El algoritmo MD5 es una función de cifrado tipo hash que acepta una cadena de texto como entrada, y devuelve un número de 128 bits. Las ventajas de este tipo de algoritmos son la imposibilidad (computacional) de reconstruir la cadena original a partir del resultado, y también la imposibilidad de encontrar dos cadenas de texto que generen el mismo resultado.

Esto nos permite usar el algoritmo para transmitir contraseñas a través de un medio inseguro. Simplemente se cifra la contraseña, y se envía de forma cifrada. En el punto de destino, para comprobar si el password es correcto, se cifra de la misma manera y se comparan las formas cifradas.

Su codificación de 128 bits es representada típicamente como un número de 32 dígitos hexadecimal.

Un ejemplo a continuación de una codificación de MD5:

En MD5 ("**esto si es una prueba de MD5**"):

Ahora vemos su Hash de salida= e99008846853ff3b725c27315e469fbc

Pero veamos lo complejo que puede llegar a ser, con tan solo cambiar una letra del mensaje original arroja un Hash de salida muy diferente;

En MD5 ("**esto no es una prueba de MD5**")

Ahora vemos su Hash de salida= dd21d99a468f3bb52a136ef5beef5034

El simple hecho de codificar un espacio vacío también da como resultado un Hash de salida complejo al igual que los antes vistos;

En MD5 (" ")

Ahora vemos su Hash de salida= d41d8cd98f00b204e9800998ecf8427e

Si se tiene un sistema de usuarios y queremos proteger las contraseñas para prevenir posibles vulnerabilidades en el servidor, es una medida eficaz encriptar las contraseñas, como se hace con el MD5, de manera que si alguien puede acceder a ellas no pueda ver la contraseña.

2.2.11. Gestor de Base de Datos

Es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. Los usuarios pueden acceder a la información usando herramientas específicas de interrogación y de generación de informes, o bien mediante aplicaciones al efecto.

Estos sistemas también proporcionan métodos para mantener la integridad de los datos, para administrar el acceso de usuarios a los datos y para recuperar la información si el sistema se corrompe. Permiten presentar la información de la base de datos en variados formatos. La mayoría incluyen un generador de informes. También pueden incluir un módulo gráfico que permita presentar la información con gráficos y tablas.

Hay muchos tipos distintos según cómo manejen los datos y muchos tamaños distintos de acuerdo a si operan en computadoras personales y con poca memoria o grandes sistemas que funcionan en mainframes con sistemas de almacenamiento especiales.

Generalmente se accede a los datos mediante lenguajes de interrogación, lenguajes de alto nivel que simplifican la tarea de construir las aplicaciones. También simplifican la interrogación y la presentación de la información. Un SGBD permite controlar el acceso a los datos, asegurar su integridad, gestionar el acceso concurrente a ellos, recuperar los datos tras un fallo del sistema y hacer copias de seguridad. Las bases de datos y los sistemas para su gestión son esenciales para cualquier área de negocio, y deben ser gestionados con esmero.

Postgresql

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia BSD.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados pororganizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilospara

garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

Este gestor de base de datos también utiliza funciones y/o procedimientos almacenados (stored procedures) en numerosos lenguajes de programacion, entre ellos PL/pgSQL, del cual hablaremos a continuación:

PL/pgSQL

PL/pgSQL es un lenguaje procedimental cargable para el sistema de base de datos PostgreSQL. Los objetivos propuestos para PL/pgSQL consisten en crear un lenguaje procedimental cargable que tenga las siguientes características:

- Que pueda ser usado para crear funciones y procedimientos disparadores.
- Adicione estructuras de control al lenguaje SQL.
- Sea capaz de realizar cálculos complejos.
- Herede todos los tipos, las funciones y los operadores definidos por el usuario.
- Pueda ser definido como confiable (trusted) por el servidor.
- Sea fácil de usar.

Excepto por las conversiones de entrada/salida y las funciones de cálculo para los tipos definidos por el usuario, todo lo que puede definirse por medio de funciones en el lenguaje C pueden definirse también con PL/pgSQL. Es posible crear funciones computacionales condicionales complejas que pueden ser usadas posteriormente para definir operadores o usarlas en expresiones asociadas a los índices.

SQL es el lenguaje que PostgreSQL y la mayoría de la bases de datos relacionales usan como lenguaje de consulta. Es portable y fácil de aprender. Pero cada sentencia SQL debe ser ejecutada individualmente por el servidor de la base de datos. Esto significa que su aplicación cliente debe enviar cada consulta al servidor de la base de datos, esperar a que sea procesada, recibir los resultados, hacer algunos cálculos, y enviar después otras consultas al servidor. Todo esto implica la comunicación entre procesos y puede también implicar una sobrecarga a la red si su cliente se encuentra en una máquina diferente a la del servidor de la base de datos. Con PL/pgSQL se puede agrupar un bloque de cálculos y una serie de consultas dentro del servidor de la base de datos, obteniendo de esta manera el poder de un lenguaje procedimental y la facilidad de uso de SQL, pero ahorrando una gran cantidad de tiempo debido a que no tiene la sobrecarga de la comunicación completa cliente/servidor. Esto puede aumentar el desempeño de una manera considerable. PL/pgSQL usa todos los tipos de datos, operadores y funciones de SQL.

CAPÍTULO 3. REQUERIMIENTOS DEL SISTEMA

Metodología RUP:

La metodología RUP es un proceso de ingeniería de software que suministra un enfoque para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta y de mayor calidad para satisfacer las necesidades de los usuarios que tienen un cumplimiento al final dentro de un limite de tiempo y presupuesto previsible. Maneja el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable (Inicio, Elaboración, Construcción y Transición) las cuales detallan las actividades a realizar para el desarrollo de software para obtener la explicación paso a paso de nuestro sistema.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. También se conoce por este nombre al software, también desarrollado por Rational, que incluye información entrelazada de diversos artefactos y descripciones de las diversas actividades y se basa en casos de uso que nos permite describir los requerimientos del software y está orientada a la arquitectura del sistema para una mejor documentación teniendo como base a UML (United Modeling Languaje).

En esta metodología se presentan distintos tipos de planificación con los que se controla el desarrollo del software, a través de un predefinido esquema de escabilidad y gestión de riesgos, se pueden conocer previamente problemas y fallas previamente y corregirlos.

Fase de Inicio

La panadería "Dos estrellas" Es una microempresa dedicada a la producción y venta de pan y otros a diferentes pequeños distribuidores además de clientes finales que realizan pedidos a diario.

En la actualidad el almacén se controla de manera manual, el principal objetivo es automatizar el proceso de logística para lograr una mejor administración de los recursos utilizados en la elaboración del producto final de una forma rápida logrando un efecto positivo en el servicio a favor de los clientes finales y la administración de la empresa.

Funciones:

Son funciones del almacén de la panadería DOS ESTRELLAS:

- Registrar cada uno de los insumos del almacén y sus cantidades
- Verificar los productos que no cumplen con la cantidad mínima para estar en almacén y coordinar la compra de los mismos inmediatamente.

3.1 REQUERIMIENTOS FUNCIONALES.

- Registrar todos y cada uno de las existencias en almacén para la producciones de pan y demás productos finales que elabore la panadería.

- Modificar y eliminar insumos según sea el requerimiento.
- Registrar las salidas de stock según la cantidad de producción diaria (mayormente 2 veces al día, mañana y tarde.)
- Al final de la producción, ingresar todos los productos finales que han sido elaborados en la jornada.
- Registrar clientes y proveedores asiduos de la empresa, así como modificar y eliminar según cual sea el requerimiento.
- Generar órdenes de compra de las existencias que estén en su cantidad mínimo o por debajo de estas para no quedarse sin stock para la producción.
- Registrar pedidos de clientes subdistribuidores según la producción diaria.
- Alertar cuando los insumos de almacén se encuentren en su cantidad mínima para realizar la adquisición de los mismos,

3.2. REQUERIMIENTOS NO FUNCIONALES

- La interfaz del sistema deberá mostrar una apariencia amigable y entendible para el usuario
- El sistema contará con un manual de ayuda que permita a los usuarios tener conocimientos sobre los aspectos generales del software además de su funcionamiento.
- El sistema deberá ser desarrollado en JAVA usando el gestor de base de datos Postgresql que son herramientas de software libre, lo que permitirá al usuario, que no cuenta con un presupuesto grande, desarrollar su sistema a un bajo costo.

CAPÍTULO 4.- Análisis y Diseño

4.1. ELABORACIÓN DE DIAGRAMAS DE CASOS DE USO

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un caso de uso se denominan actores.

En el contexto de ingeniería del software, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones.

Se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo.

Los más comunes para la captura de requisitos funcionales, especialmente con el desarrollo del paradigma de la programación orientada a objetos, donde se originaron, si bien puede utilizarse con resultados igualmente satisfactorios con otros paradigmas de programación.

4.1.1 Módulo Gestión de almacén

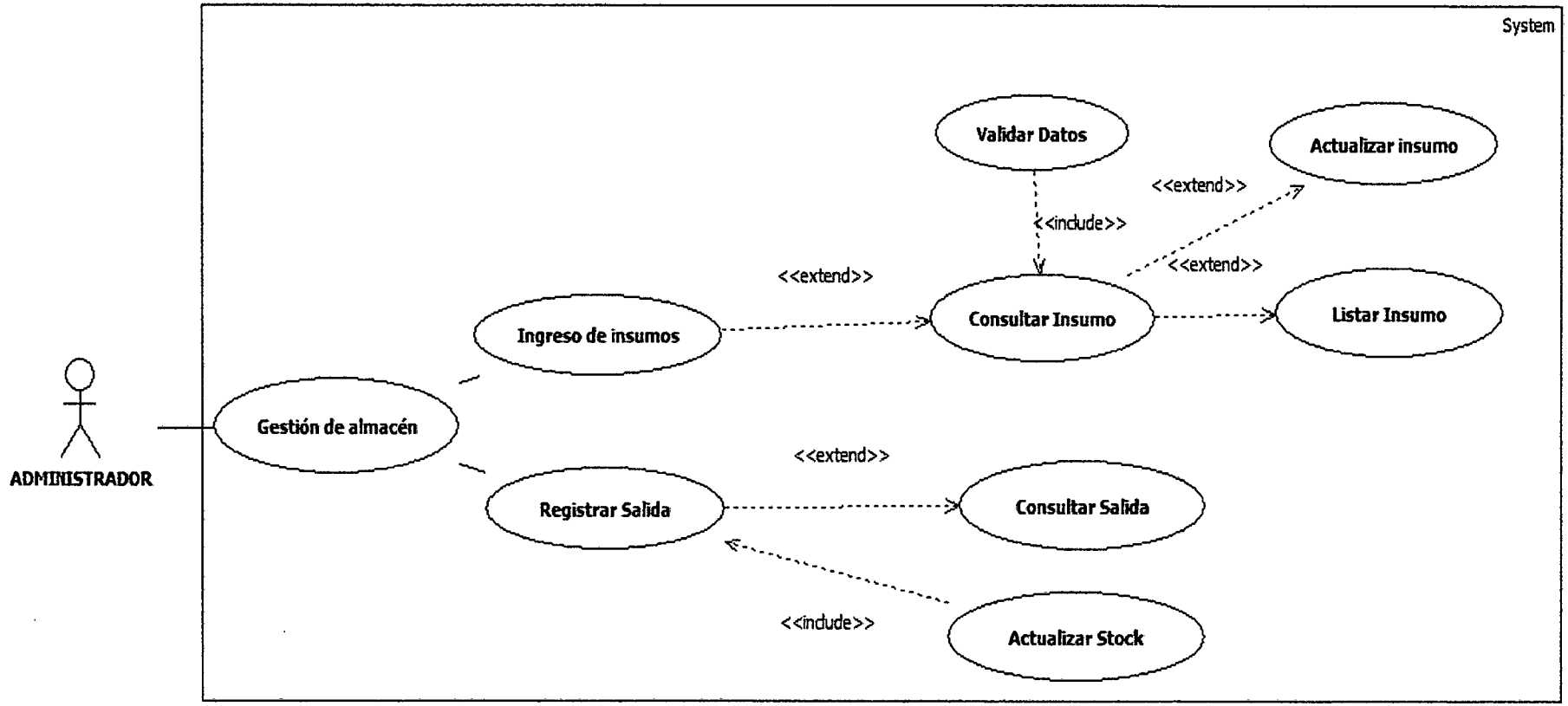


Figura 4.1.1.1: Casa de Uso Gestión de almacén

- CU 1: INGRESAR INSUMO

Nombre de Caso de Uso	INGRESAR INSUMO
Actor	Administrador
Descripción	El caso de uso registrará un nuevo insumo con sus respectivas características.
Precondición	Iniciar sesión
Secuencia normal:	
1. El Sistema muestra formulario para ingresar Insumo 2. El Usuario Ingresa datos del Insumo 3. El sistema valida datos del insumo. 4. El sistema lanza mensaje de alerta 5. Usuario confirma la orden. 6. El sistema guarda datos del insumo.	

Tabla 4.1.1.1: CU Ingresar Insumo

- CU2: CONSULTAR INSUMO

Nombre de Caso de Uso	CONSULTAR INSUMO
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de un insumo ya ingresado, proporcionando facilidades para encontrar dicha materia prima según diferentes atributos de consulta.
Precondición	Iniciar sesión, ingresar insumo
Secuencia normal:	

1. El Usuario elije opción de consulta de insumos 2. El sistema presenta el formulario para consulta 3. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso 4. El Sistema muestra información de insumo.

Tabla 4.1.1.2: CU Consultar Insumo

• CU3: ACTUALIZAR INSUMO

Nombre de Caso de Uso	ACTUALIZAR INSUMO
Actor	Administrador
Descripción	El caso de uso Describe el proceso para actualizar los datos del insumo según requerimientos y campos activos para su modificación.
Precondición	Iniciar sesión, existencia del insumo
Secuencia normal:	
1. El Sistema muestra información completa del insumo 2. El Usuario elije opción a modificar. 3. El sistema valida los datos ingresados 4. El sistema lanza mensaje de alerta. 5. Usuario confirma los cambios 6. El sistema guarda los datos del insumo.	

Tabla 4.1.1.3: CU Actualizar Insumo

• CU4: REGISTRAR SALIDA

Nombre de Caso de Uso	REGISTRAR SALIDA
Actor	Administrador

Descripción	El caso de uso registrará la salida de la lista de productos que se necesitaran para la
Precondición	
Secuencia normal:	
1. El Sistema muestra formulario con productos disponibles en almacén 2. El Usuario elije insumos para dar salida. 3. El sistema valida los datos ingresados 4. El usuario confirma salida de lista de insumos 5. El sistema actualiza el stock de insumos. 6. El sistema guarda la lista de salida de insumos.	

Tabla 4.1.1.4: CU Registrar salida

- CU5: CONSULTAR SALIDA

Nombre de Caso de Uso	CONSULTAR SALIDA
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de la salida de insumos, según fecha y hora del día para llevar un control de la materia prima utilizada por los empleados.
Precondición	Iniciar sesión, ingresar salida
Secuencia normal:	
1. El Usuario elije opción de consulta de salida. 2. El sistema presenta el formulario para consulta 3. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso 4. El Sistema muestra información de salida registrada.	

Tabla 4.1.1.5: CU Consultar salida

- CU6: ACTUALIZAR STOCK

Nombre de Caso de Uso	ACTUALIZAR STOCK
Actor	Administrador
Descripción	El caso describe el proceso para modificar el atributo stock de cada uno de los atributos.
Precondición	
Secuencia normal:	
1. El sistema busca insumo por código. 2. El sistema modifica el atributo CANTIDAD 3. El sistema guarda nuevo valor para el atributo	

Tabla 4.1.1.5: CU Actualizar Stock

4.1.2. Modulo Gestión de Orden de Compra

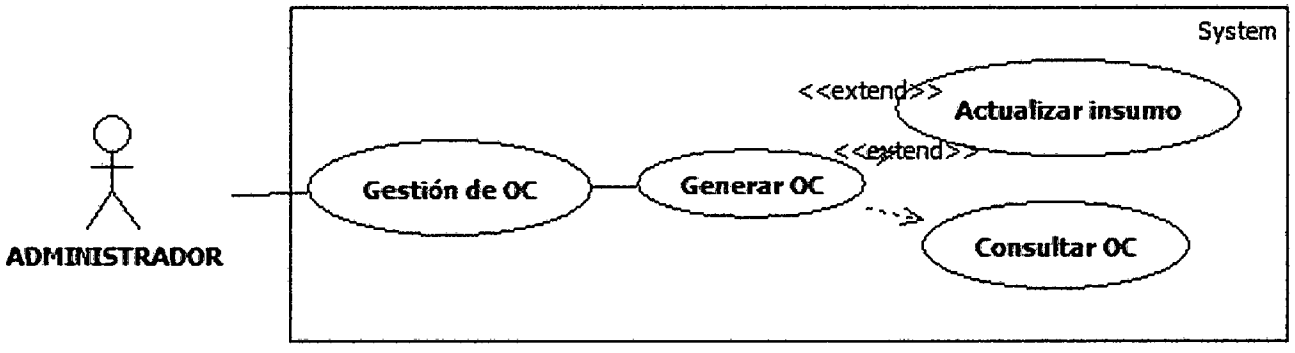


Figura 4.1.2.1 Caso de Uso Gestión de Orden de Compra

- CU7: GENERAR ORDEN DE COMPRA

Nombre de Caso de Uso	GENERAR ORDEN DE COMPRA
Actor	Administrador

Descripción	El caso de uso describe el proceso para crear una orden de compra en la cual se incluirán todos los productos que estén por debajo de su existencia mínima en el almacén.
Precondición	Iniciar sesión
Secuencia normal:	
1. El usuario elige la opción GENERAR ORDEN DE COMPRA 2. El sistema muestra formulario. 3. El usuario elegirá los productos a incluir en la OC según el proveedor. 4. El sistema lanzará un mensaje de alerta al usuario. 5. El usuario confirma datos de la orden de compra. 6. El sistema guarda datos.	

Tabla 4.2.1.1 CU Generar Orden de compra

- CU8: CONSULTAR ORDEN DE COMPRA

Nombre de Caso de Uso	CONSULTAR ORDEN DE COMPRA
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de la orden de compra generada.
Precondición	Iniciar sesión, ingresar orden de compra
Secuencia normal:	
1. El Usuario elije opción de consulta de orden de compra 2. El sistema presenta el formulario para consulta 3. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso 4. El Sistema muestra información de orden de compra.	

Tabla 4.2.1.2 CU Consultar Orden de compra

4.1.3.- Módulo Gestión de clientes y proveedores

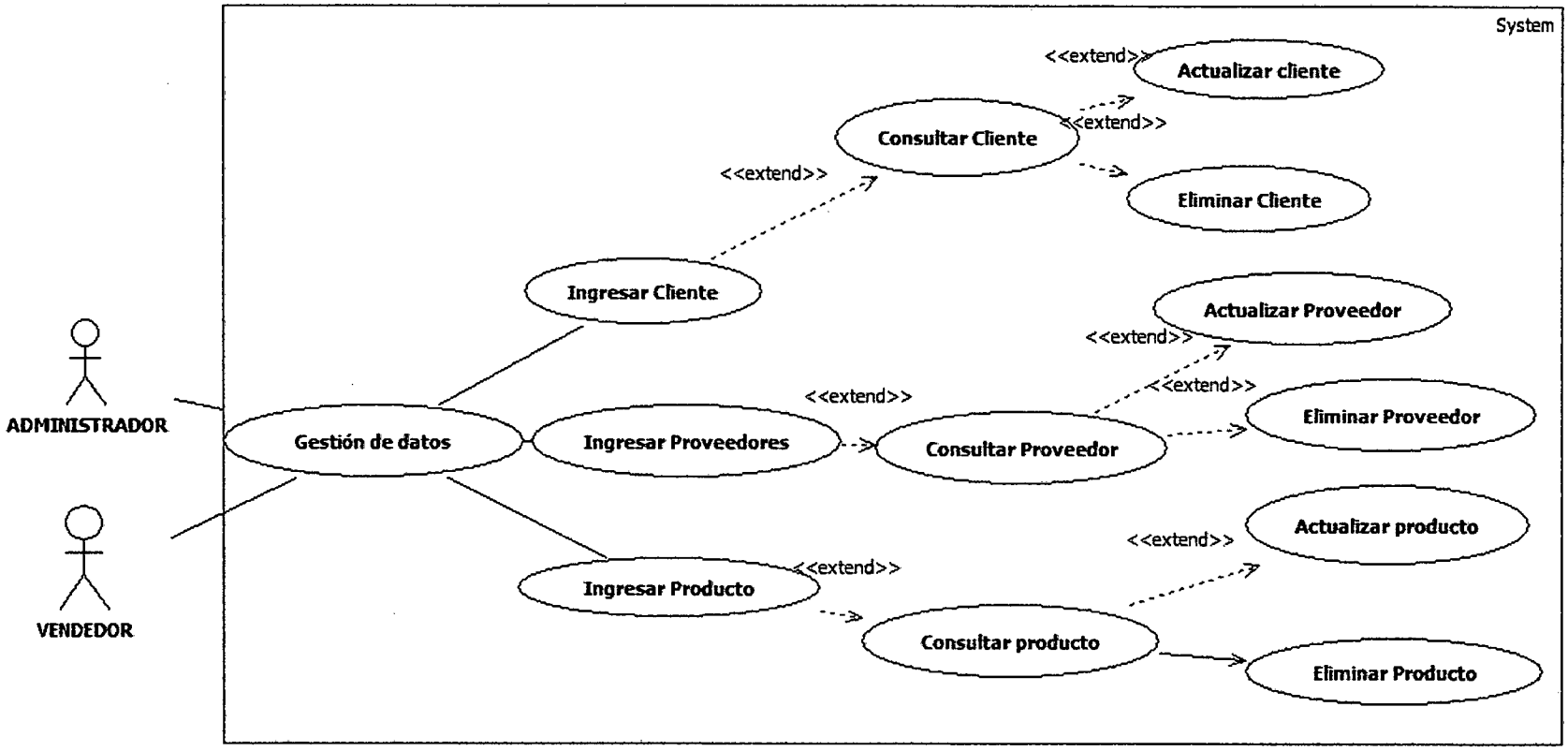


Figura 4.1.3.1. Caso de Uso Gestión de Datos

- CU9: INGRESAR CLIENTE

Nombre de Caso de Uso	INGRESAR CLIENTE
Actor	Administrador
Descripción	El caso de uso registrará un nuevo cliente de subdistribución del producto final de la panadería.
Precondición	Iniciar sesión
Secuencia normal:	
1. El Sistema muestra formulario para ingresar Cliente 2. El Usuario Ingresa datos del cliente 3. El sistema valida datos del cliente 4. El sistema lanza mensaje de alerta 5. El Usuario confirma 6. El sistema guarda datos del cliente.	

Tabla 4.1.3.1. CU Ingresar

- CU10: CONSULTAR CLIENTE

Nombre de Caso de Uso	CONSULTAR CLIENTE
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de un cliente registrado.
Precondición	Iniciar sesión, ingresar cliente
Secuencia normal:	
5. El Usuario elije opción de consulta de cliente 6. El sistema presenta el formulario para consulta 7. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso 8. El Sistema muestra información de cliente.	

Tabla 4.1.3.2. CU Consultar

- CU11: ACTUALIZAR CLIENTE

Nombre de Caso de Uso	ACTUALIZAR CLIENTE
Actor	Administrador
Descripción	El caso de uso guardará datos modificados del cliente.
Precondición	Iniciar sesión, existencia del cliente.
Secuencia normal:	
1. El Sistema muestra información completa del cliente 2. El Usuario elije opción a modificar. 3. El sistema valida los datos ingresados 4. El sistema lanza mensaje de alerta. 5. Usuario confirma los cambios 6. El sistema guarda los datos del cliente.	

Tabla 4.1.3.3. CU Actualizar

- CU12: INGRESAR PROVEEDOR

Nombre de Caso de Uso	INGRESAR PROVEEDOR
Actor	Administrador
Descripción	El caso de uso registrará un nuevo proveedor asiduo del administrador de la organización.
Precondición	Iniciar sesión
Secuencia normal:	

1. El Sistema muestra formulario para ingresar proveedor 2. El Usuario Ingresa datos del proveedor 3. El sistema valida datos del proveedor 4. El sistema lanza mensaje de alerta 5. El Usuario confirma 6. El sistema guarda datos del proveedor.

Tabla 4.1.3.4. CU Ingresar Proveedor

- CU13: CONSULTAR PROVEEDOR

Nombre de Caso de Uso	CONSULTAR PROVEEDOR
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de un proveedor registrado.
Precondición	Iniciar sesión, ingresar proveedor
Secuencia normal:	
1. El Usuario elije opción de consulta de proveedor 2. El sistema presenta el formulario para consulta 3. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso 4. El Sistema muestra información de proveedor.	

Tabla 4.1.3.4. CU Consultar Proveedor

- CU14: ACTUALIZAR PROVEEDOR

Nombre de Caso de Uso	ACTUALIZAR PROVEEDOR
Actor	Administrador
Descripción	El caso de uso guardará datos modificados del proveedor
Precondición	Iniciar sesión, existencia del proveedor.
Secuencia normal:	

1. El Sistema muestra información completa del proveedor
2. El Usuario elije opción a modificar.
3. El sistema valida los datos ingresados
4. El sistema lanza mensaje de alerta.
5. Usuario confirma los cambios
6. El sistema guarda los datos del proveedor.

Tabla 4.1.3.5. CU Actualizar Proveedor

4.1.4.- Módulo Gestión de Pedidos

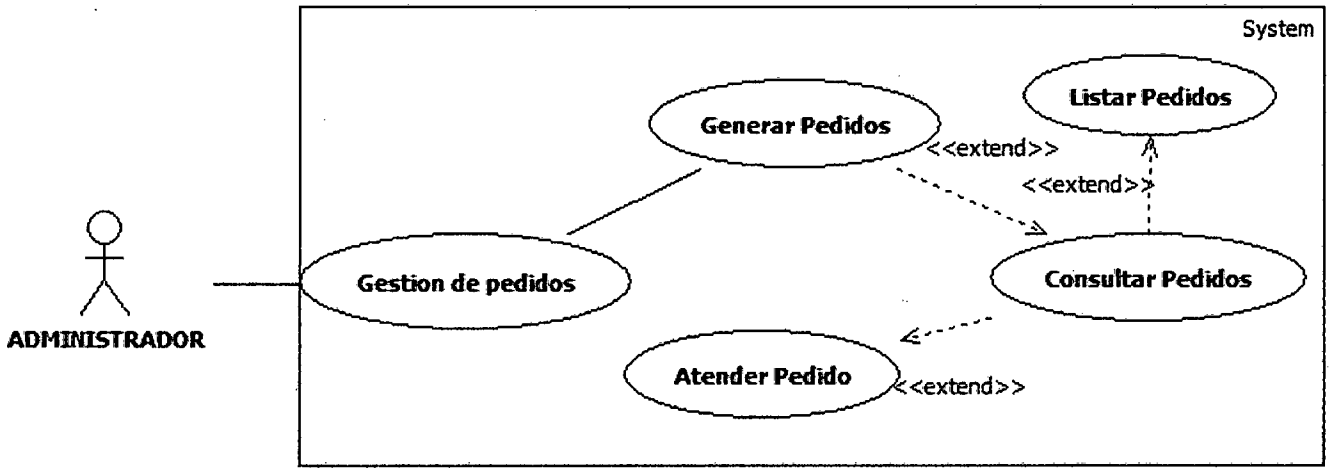


Figura 4.1.4.1 Caso de Uso Gestión de Pedidos

- CU15: GENERAR PEDIDO

Nombre de Caso de Uso	GENERAR PEDIDO
Actor	Administrador

Descripción	El caso de uso describe el registro de pedidos solicitados por clientes subdistribuidores con todos sus atributos para ser atendido con productos finales.
Precondición	Iniciar sesión
Secuencia normal:	
1. El usuario elige la opción GENERAR PEDIDO 2. El sistema muestra formulario. 3. El usuario indicará los productos que desea incluir en el pedido. 4. El sistema lanzará un mensaje de alerta al usuario. 5. El usuario confirma datos del pedido **6.** El sistema guarda datos.	

Tabla 4.1.4.1 Generar Pedido

- CU16: CONSULTAR PEDIDO

Nombre de Caso de Uso	CONSULTAR PEDIDO
Actor	Administrador
Descripción	El caso de uso describe la búsqueda de pedidos ingresados.
Precondición	Iniciar sesión, ingresar pedido.
Secuencia normal:	
1. El sistema presenta el formulario para consulta 2. El Sistema verifica parámetros - Si los datos son incorrectos vuelve a paso 2 e Informa el suceso **3.** El Sistema muestra información de pedido.	

Tabla 4.1.4.2 Consultar Pedido

4.2 ELABORACIÓN DE DIAGRAMAS DE SECUENCIA

4.2.1. MODULO ALMACEN:

INGRESAR INSUMO

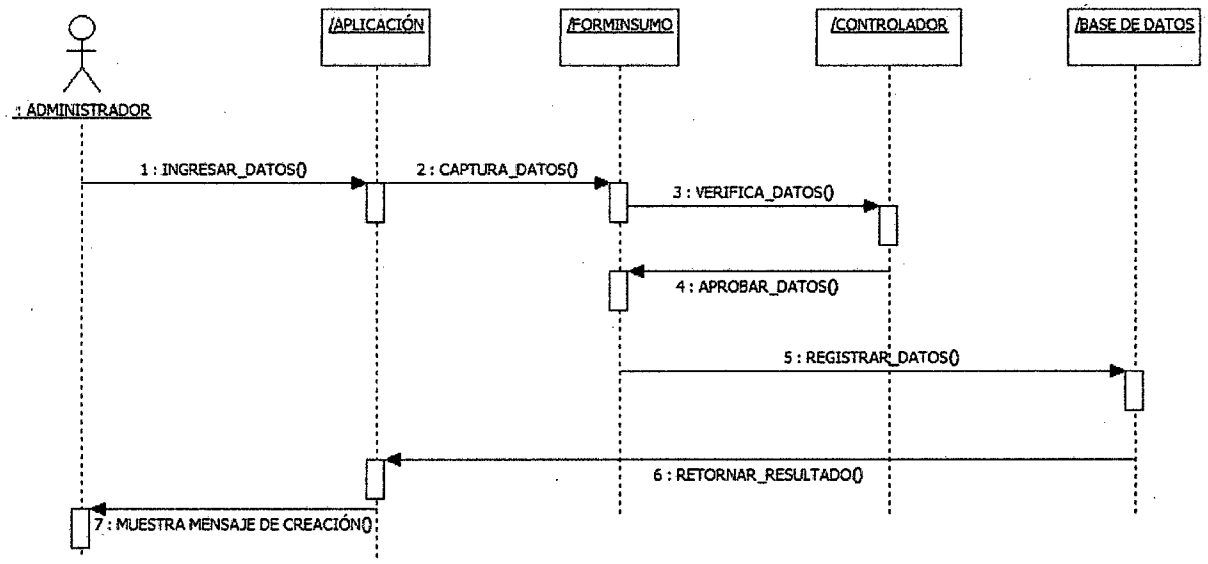


Figura 4.2.1.1 Ingresar Insumo

CONSULTAR INSUMO

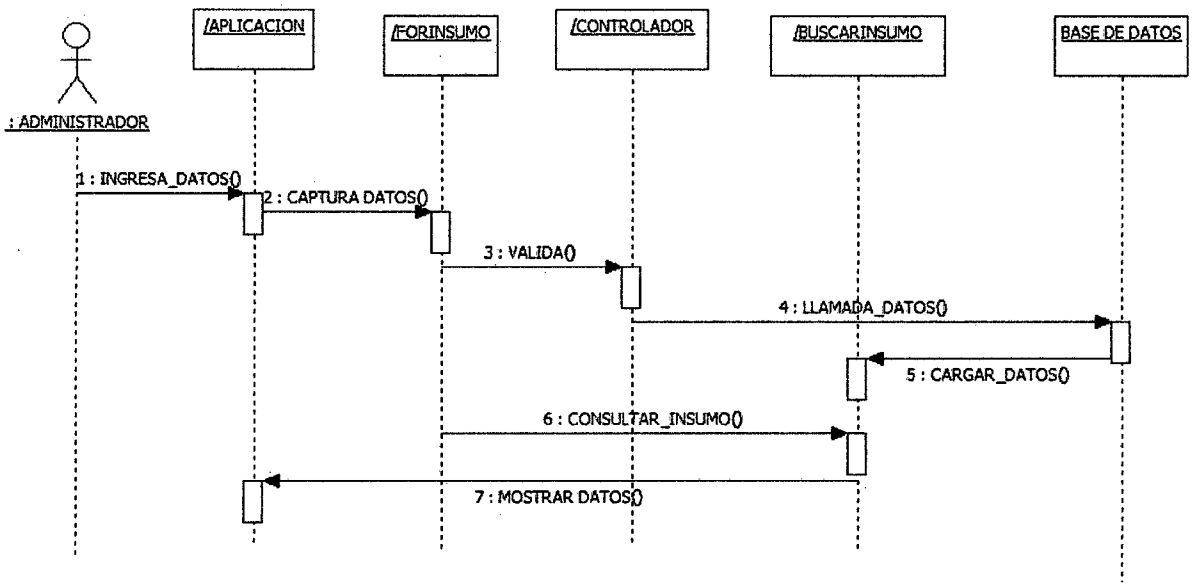


Figura 4.2.1.2 Consultar insumo

ACTUALIZAR INSUMO

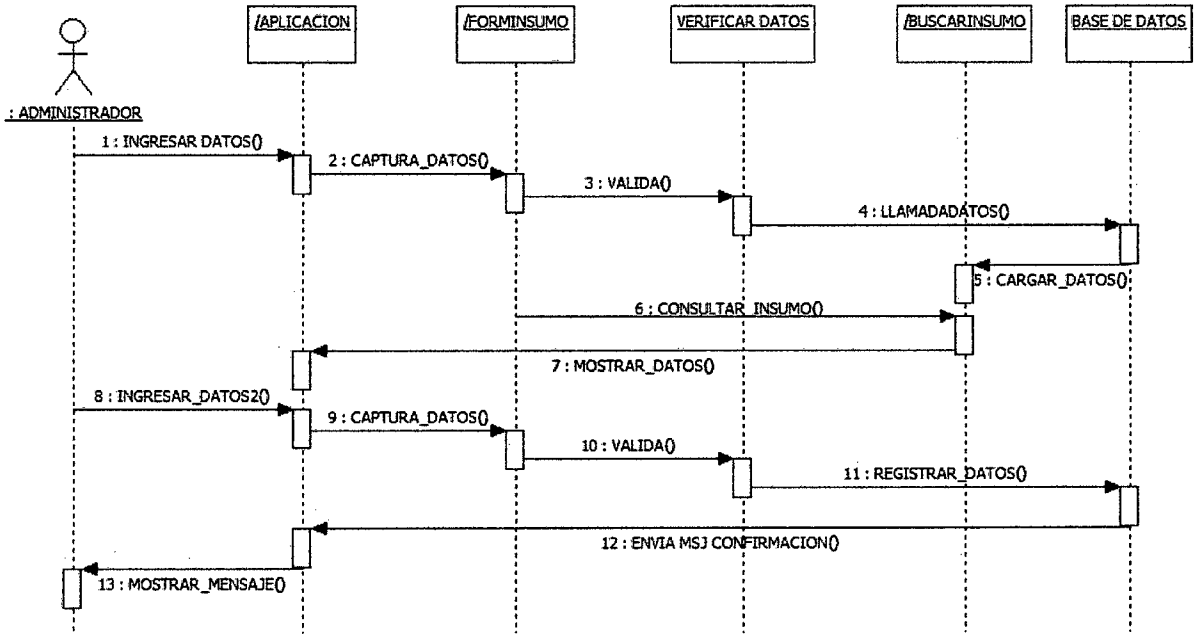


Figura 4.2.1.3 Actualizar insumo

REGISTRAR SALIDAS

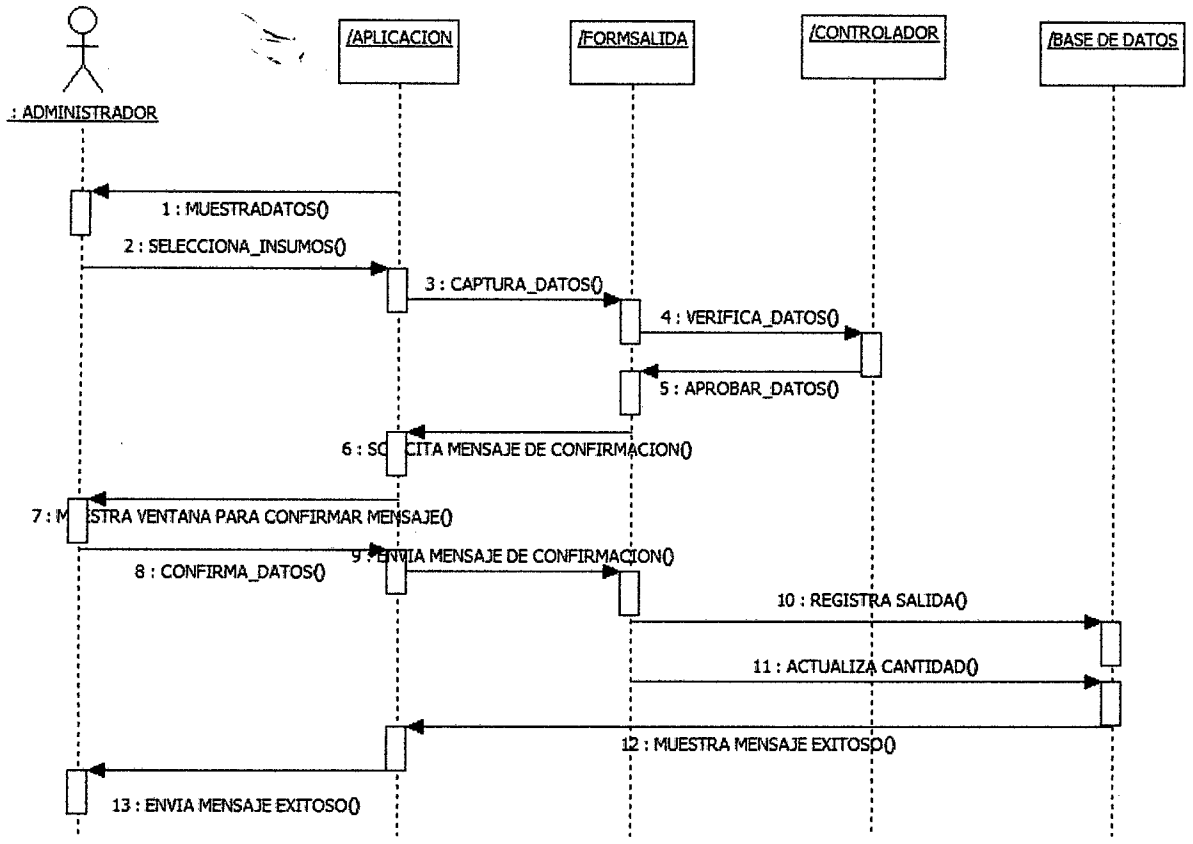


Figura 4.2.1.4 Registrar Salida

CONSULTAR SALIDA

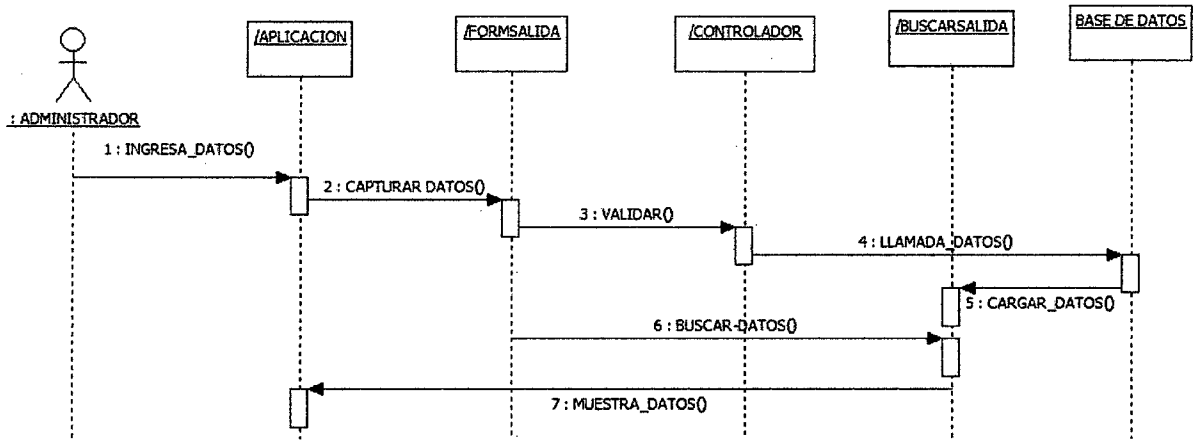


Figura 4.2.1.5 Consultar Salida

ACTUALIZAR STOCK

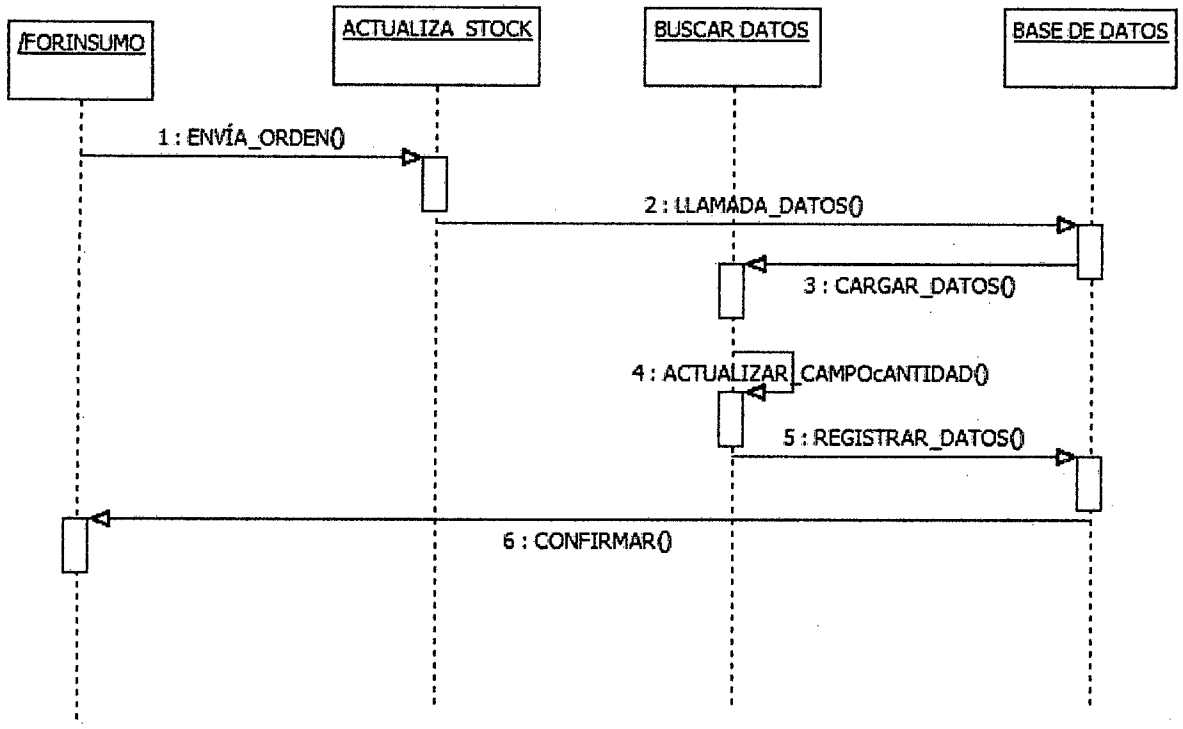


Figura 4.2.1.6 Actualizar Stock

4.2.2. MODULO COMPRAS

GENERAR OC

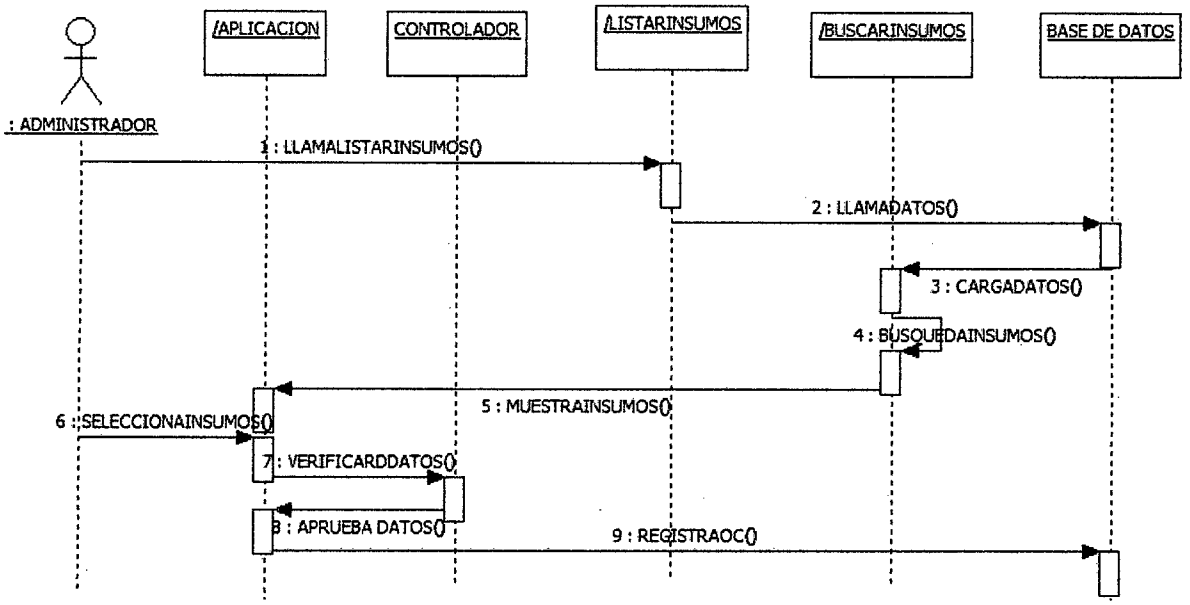


Figura 4.2.2.1 Generar OC

CONSULTAR OC

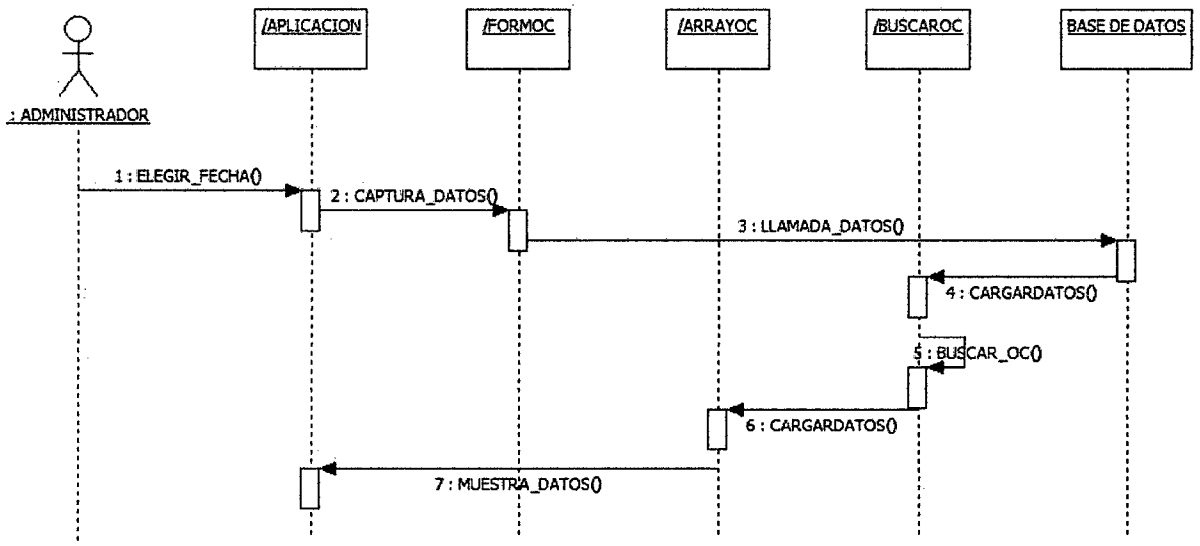


Figura 4.2.2.2 Consultar OC

4.2.3. MÓDULO ARCHIVO

INGRESAR

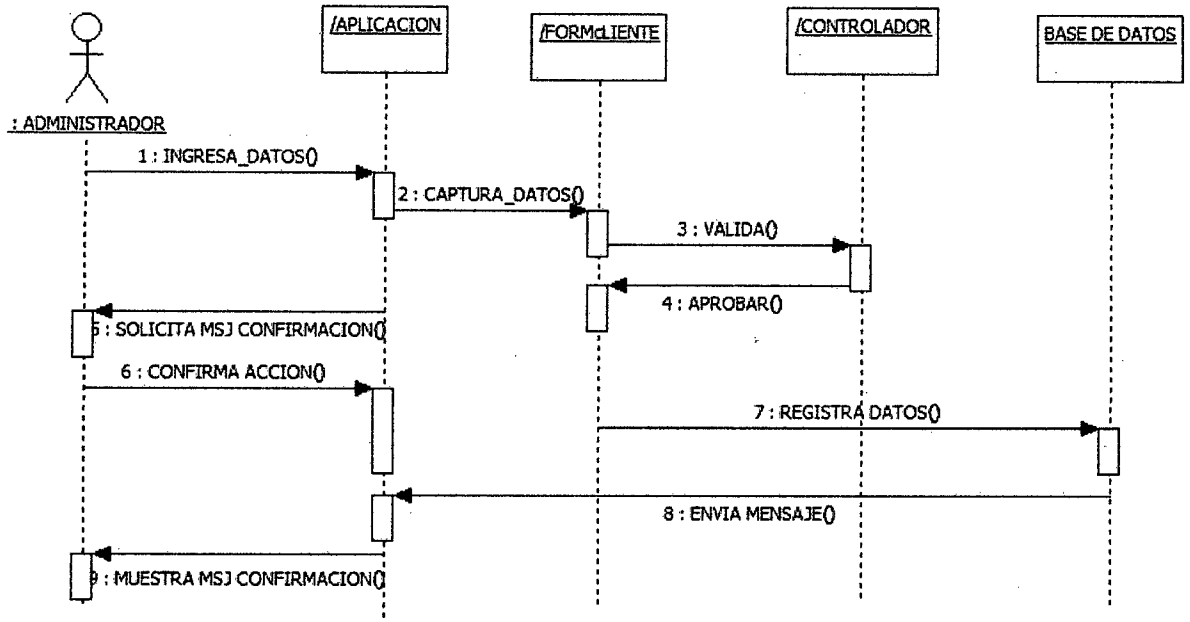


Figura 4.2.3.1 Ingresar

CONSULTAR

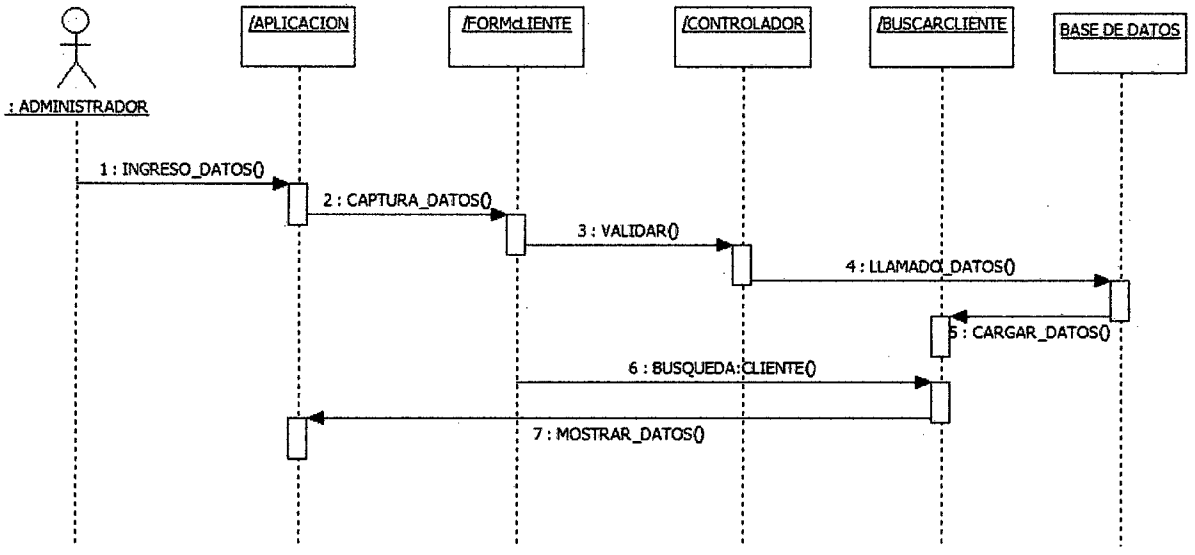


Figura 4.2.3.2 Consultar

ACTUALIZAR

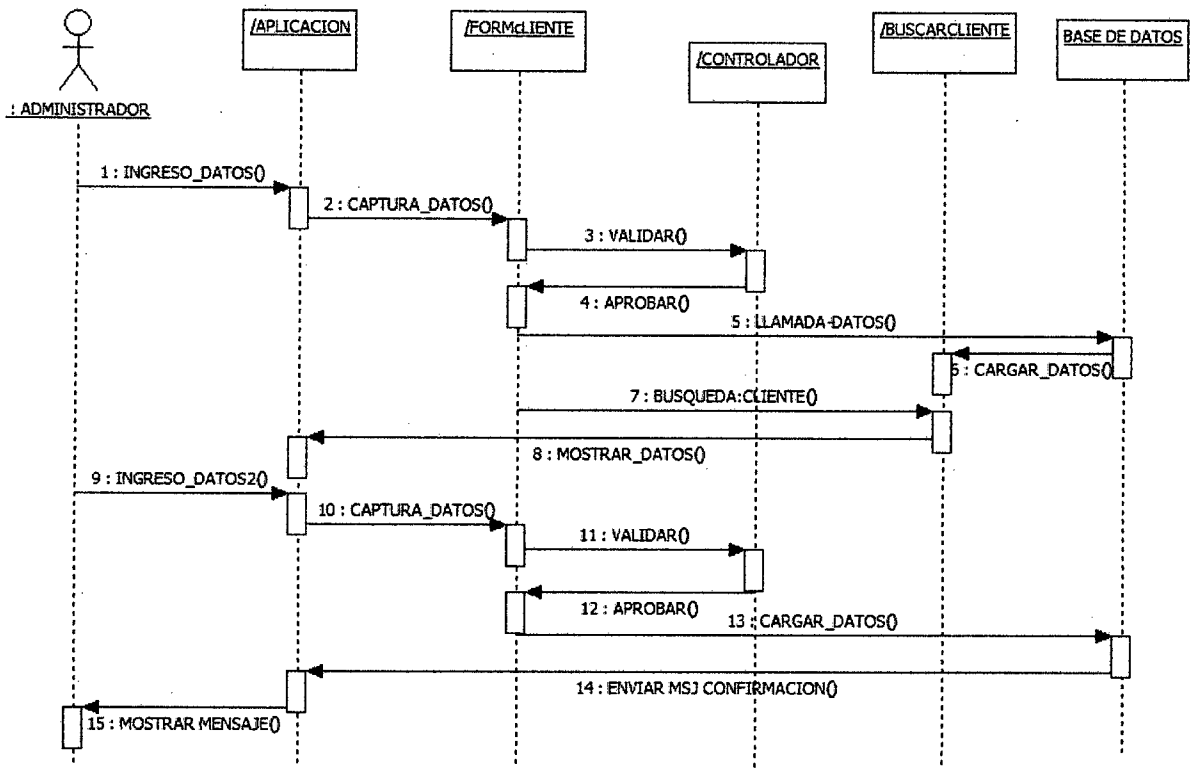


Figura 4.2.3.3 Actualizar

ELIMINAR

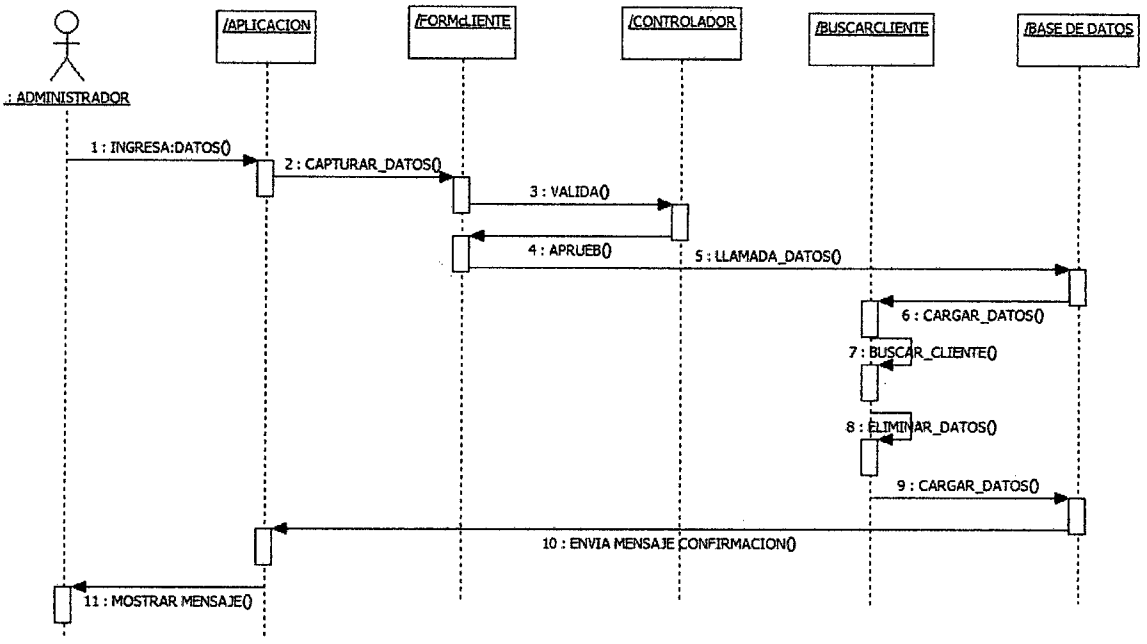


Figura 4.2.3.4 Eliminar

4.2.4. MODULO PEDIDOS

GENERAR PEDIDO

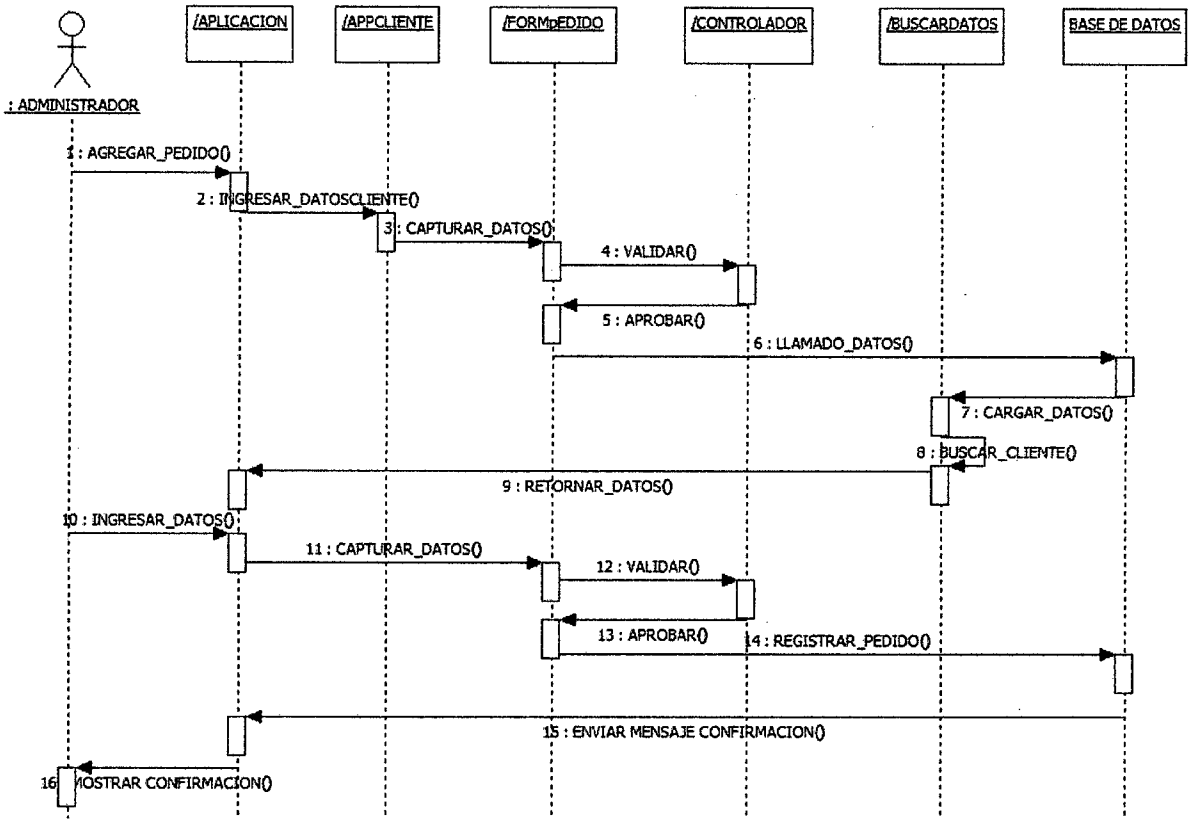


Figura 4.2.4.1 Generar Pedido

CONSULTAR PEDIDO

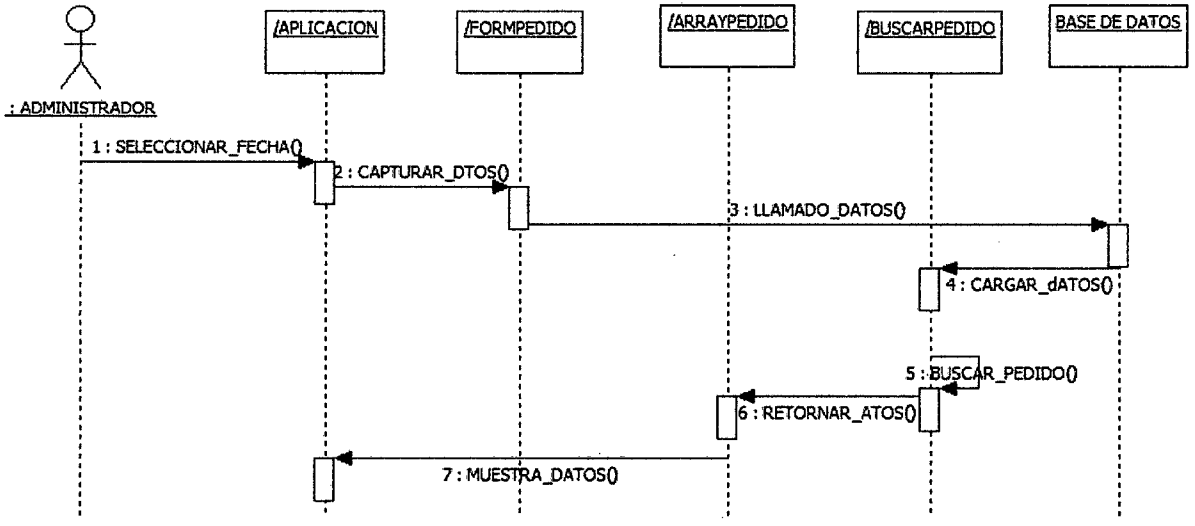
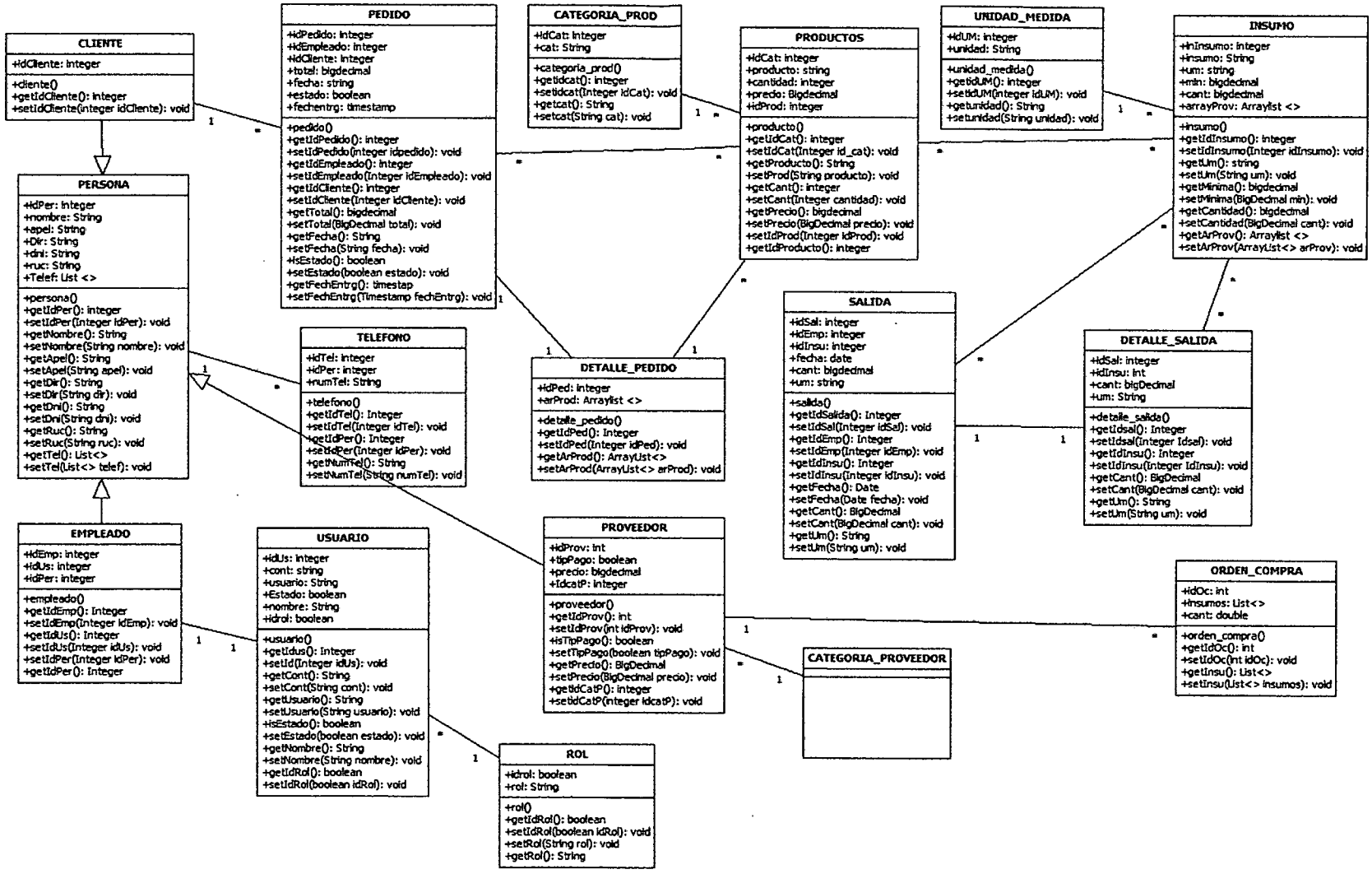


Figura 4.2.4.2 Generar Pedido

4.3. DIAGRAMA DE CLASES


CLIENTE
+idCliente: Integer
+cliente()
+getIdCliente(): integer
+setIdCliente(integer idCliente): void
PERSONA
+idPer: Integer
+nombre: String
+apel: String
+Dir: String
+dni: String
+ruc: String
+Telef: List <>
+persona()
+getIdPer(): integer
+setIdPer(Integer idPer): void
+getNombre(): String
+setNombre(String nombre): void
+getApel(): String
+setApel(String apel): void
+getDir(): String
+setDir(String dir): void
+getDni(): String
+setDni(String dni): void
+getRuc(): String
+setRuc(String ruc): void
+getTel(): List<>
+setTel(List<> telef): void
EMPLEADO
+idEmp: integer
+idUs: integer
+idPer: integer
+empleado()
+getIdEmp(): Integer
+setIdEmp(Integer idEmp): void
+getIdUs(): Integer
+setIdUs(Integer idUs): void
+setIdPer(Integer idPer): void
+getIdPer(): Integer
PEDIDO
+idPedido: integer
+idEmpleado: integer
+idCliente: integer
+total: bigdecimal
+fecha: string
+estado: boolean
+fechentrg: timestamp
+pedido()
+getIdPedido(): integer
+setIdPedido(integer idpedido): void
+getIdEmpleado(): integer
+setIdEmpleado(Integer idEmpleado): void
+getIdCliente(): integer
+setIdCliente(Integer idCliente): void
+getTotal(): bigdecimal
+setTotal(BigDecimal total): void
+getFecha(): String
+setFecha(String fecha): void
+isEstado(): boolean
+setEstado(boolean estado): void
+getFechEntrg(): timestap
+setFechEntrg(Timestamp fechEntrg): void
TELEFONO
+idTel: integer
+idPer: integer
+numTel: String
+telefono()
+getIdTel(): Integer
+setIdTel(Integer idTel): void
+getIdPer(): Integer
+setIdPer(Integer idPer): void
+getNumTel(): String
+setNumTel(String numTel): void
USUARIO
+idUs: integer
+cont: string
+usuario: String
+Estado: boolean
+nombre: String
+idrol: boolean
+usuario()
+getIdus(): Integer
+setId(Integer idUs): void
+getCont(): String
+setCont(String cont): void
+getUsuario(): String
+setUsuario(String usuario): void
+isEstado(): boolean
+setEstado(boolean estado): void
+getNombre(): String
+setNombre(String nombre): void
+getIdRol(): boolean
+setIdRol(boolean idRol): void
ROL
+idrol: boolean
+rol: String
+rol()
+getIdRol(): boolean
+setIdRol(boolean idRol): void
+setRol(String rol): void
+getRol(): String
CATEGORIA_PROD
+idCat: integer
+cat: String
+categoria_prod()
+getidcat(): integer
+setidcat(integer idCat): void
+getcat(): String
+setcat(String cat): void
DETALLE_PEDIDO
+idPed: integer
+arProd: Arraylist <>
+detalle_pedido()
+getIdPed(): Integer
+setIdPed(Integer idPed): void
+getArProd(): ArrayList<>
+setArProd(ArrayList<> arProd): void
PRODUCTOS
+idCat: integer
+producto: string
+cantidad: integer
+precio: Bigdecimal
+idProd: integer
+producto()
+getIdCat(): integer
+setIdCat(Integer id_cat): void
+getProducto(): String
+setProd(String producto): void
+getCant(): integer
+setCant(Integer cantidad): void
+getPrecio(): bigdecimal
+setPrecio(BigDecimal precio): void
+setIdProd(Integer idProd): void
+getIdProducto(): integer
PROVEEDOR
+idProv: int
+tipPago: boolean
+precio: bigdecimal
+IdcatP: integer
+proveedor()
+getIdProv(): int
+setIdProv(int idProv): void
+isTipPago(): boolean
+setTipPago(boolean tipPago): void
+getPrecio(): BigDecimal
+setPrecio(BigDecimal precio): void
+getIdCatP(): integer
+setIdCatP(integer idcatP): void
CATEGORIA_PROVEEDOR
UNIDAD_MEDIDA
+idUM: integer
+unidad: String
+unidad_medida()
+getidUM(): integer
+setidUM(integer idUM): void
+getunidad(): String
+setunidad(String unidad): void
SALIDA
+idSal: integer
+idEmp: integer
+idInsu: integer
+fecha: date
+cant: bigdecimal
+um: string
+salida()
+getIdSalida(): Integer
+setIdSal(Integer idSal): void
+getIdEmp(): Integer
+setIdEmp(Integer idEmp): void
+getIdInsu(): Integer
+setIdInsu(Integer idInsu): void
+getFecha(): Date
+setFecha(Date fecha): void
+getCant(): BigDecimal
+setCant(BigDecimal cant): void
+getUm(): String
+setUm(String um): void
INSUMO
+idInsumo: integer
+insumo: String
+um: string
+min: bigdecimal
+cant: bigdecimal
+arrayProv: Arraylist <>
+insumo()
+getIdInsumo(): integer
+setIdInsumo(Integer idInsumo): void
+getUm(): string
+setUm(String um): void
+getMinima(): bigdecimal
+setMinima(BigDecimal min): void
+getCantidad(): bigdecimal
+setCantidad(BigDecimal cant): void
+getArProv(): Arraylist <>
+setArProv(ArrayList<> arProv): void
DETALLE_SALIDA
+idSal: integer
+idInsu: int
+cant: bigDecimal
+um: String
+detalle_salida()
+getIdsal(): Integer
+setIdsal(Integer Idsal): void
+getIdInsu(): Integer
+setIdInsu(Integer IdInsu): void
+getCant(): BigDecimal
+setCant(BigDecimal cant): void
+getUm(): String
+setUm(String um): void
ORDEN_COMPRA
+idOc: int
+insumos: List<>
+cant: double
+orden_compra()
+getIdOc(): int
+setIdOc(int idOc): void
+getInsu(): List<>
+setInsu(List<> insumos): void

4.4. DISEÑO DE LA INTERFAZ

4.4.1. Ingreso al sistema:

Para ingresar al sistema se debe contar con un usuario y contraseña el cual es creado por el administrador el que tendrá todos los accesos al sistema.

Para el ingreso al sistema de Panadería aparecerá la siguiente ventana para el ingreso del usuario y contraseña correspondiente para la acción.

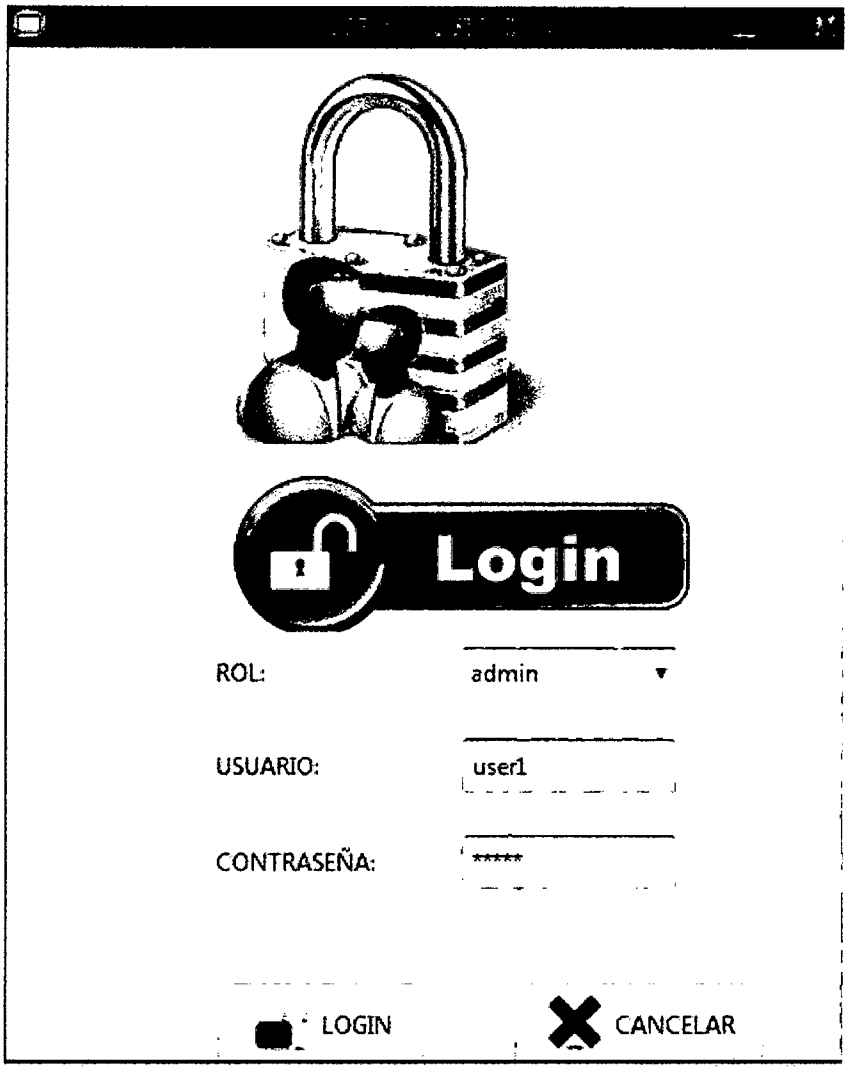
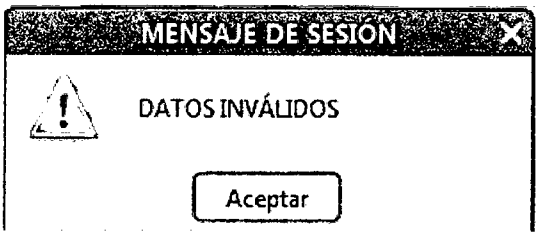


Figura 4.4.1.1 Ingreso al sistema

Si no se cuenta con un registro o los datos no son válidos se mostrará el siguiente mensaje:



4.4.2 Menú principal

Como se ha mencionado, la aplicación se ha diseñado para el ingreso de datos por parte del usuario a la Base de datos.

Si hemos ingresado los datos correctos de usuario aparecerá la pantalla del menú principal de la aplicación, en ella se ubican los módulos ya mencionados como:

- Archivo
- Pedido
- Almacén
- Compras

Que permiten ingresar a las diferentes acciones que se pueden realizar en el sistema.

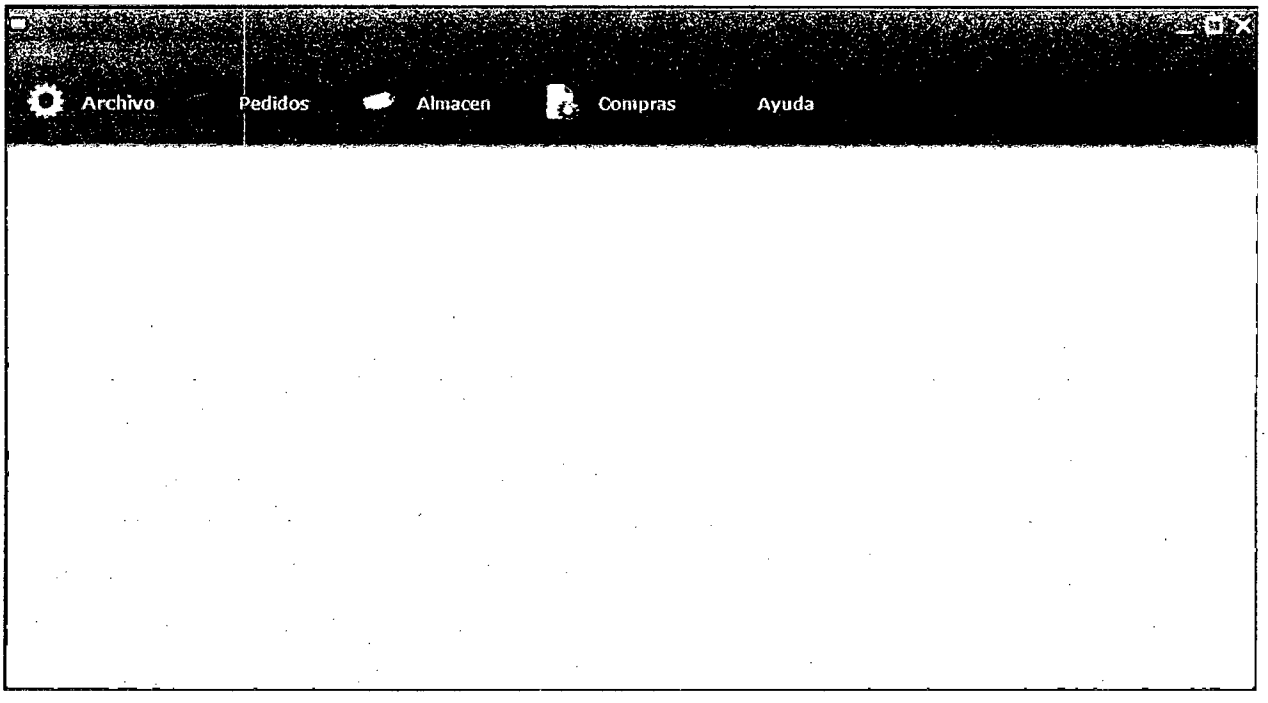


Figura 4.4.2.1 Menú Principal

4.4.3 Archivo

Ingresa los clientes, proveedores y productos finales preparados en la panadería, además de los procesos de modificación eliminación y búsqueda de ellos.

En este módulo encontraremos las siguientes pestañas:

- Mantenimiento Cliente
- Mantenimiento Producto
- Mantenimiento Proveedor
- Buscar Cliente
- Buscar Proveedor
- Mantenimiento Usuarios
- Listar Productos

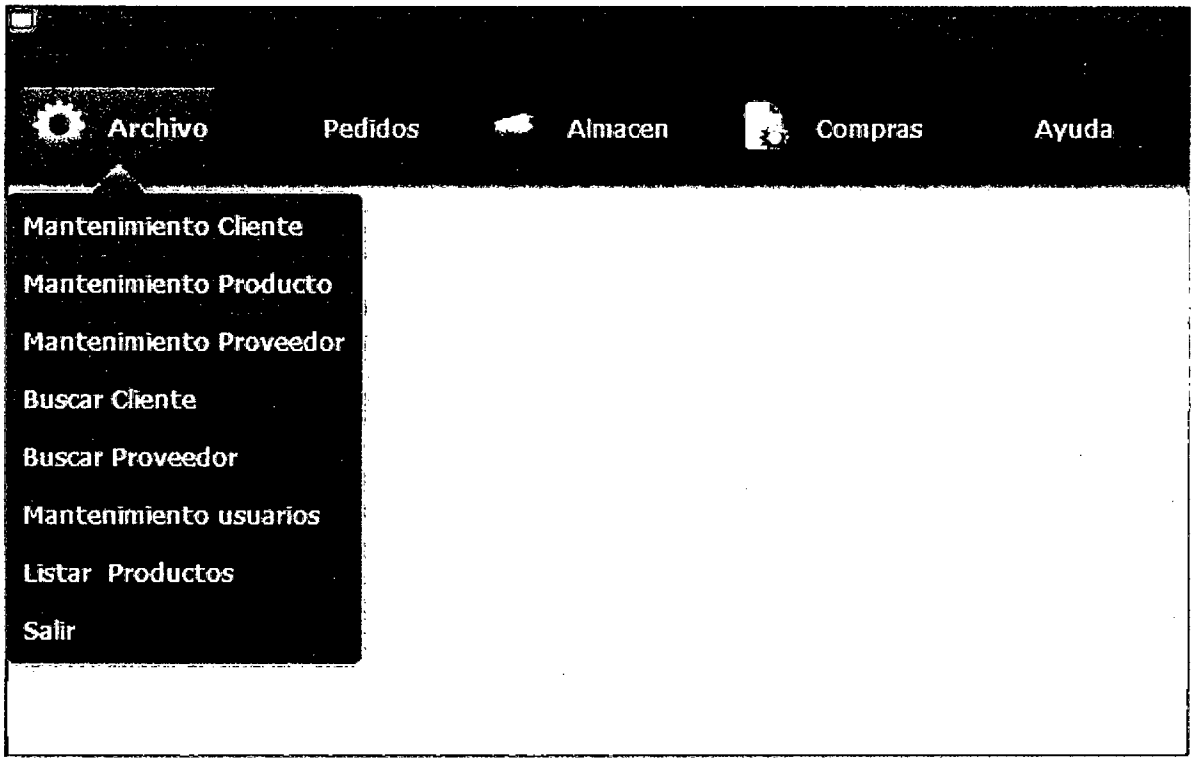


Figura 4.4.3.1 Archivo

Mantenimiento de cliente

En esta pestaña permite ingresar, modificar, eliminar clientes, validando que los datos requeridos para la próximas operaciones se hayan ingresado correctamente y completos.

Al ingresar todos los datos se presionará el botón GUARDAR, estos se almacenarán en la Base de datos y aparecerán automáticamente en la Tabla que se muestra en la parte inferior de la ventana en la cual se mostrarán de 10 en 10.

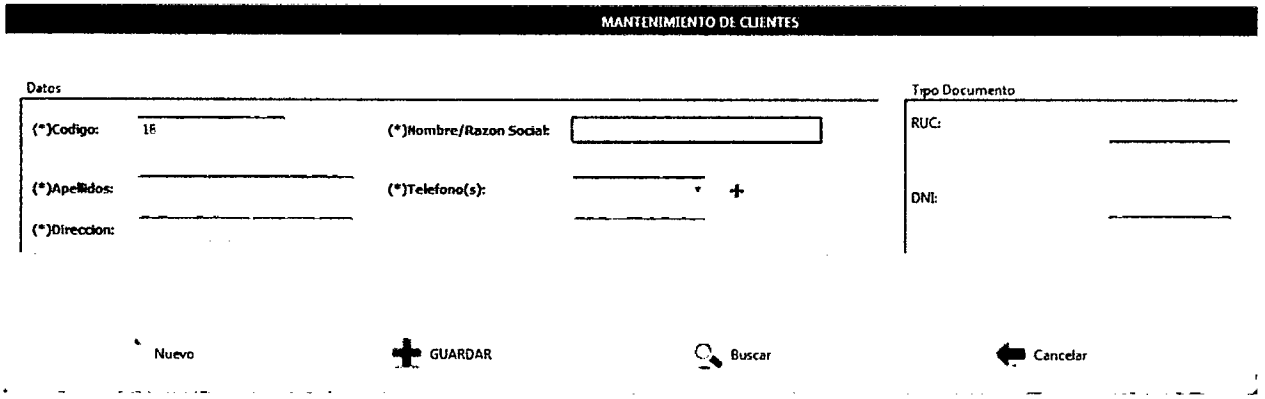


Figura 4.4.3.2. Clientes

Buscar cliente:

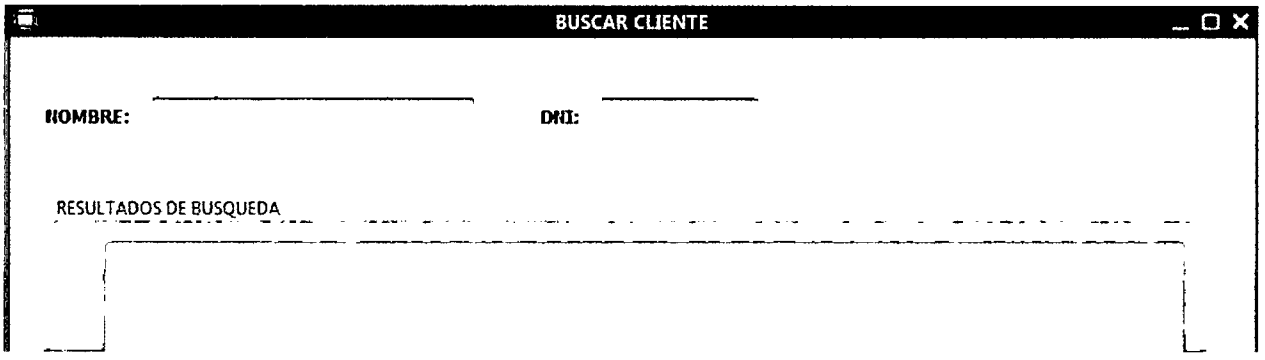


Figura 4.4.3.3 buscar Cliente

Mantenimiento de Producto

Registra, y busca la producción diaria según lo ingresado diariamente para su control en la atención de los pedidos de los productos finales

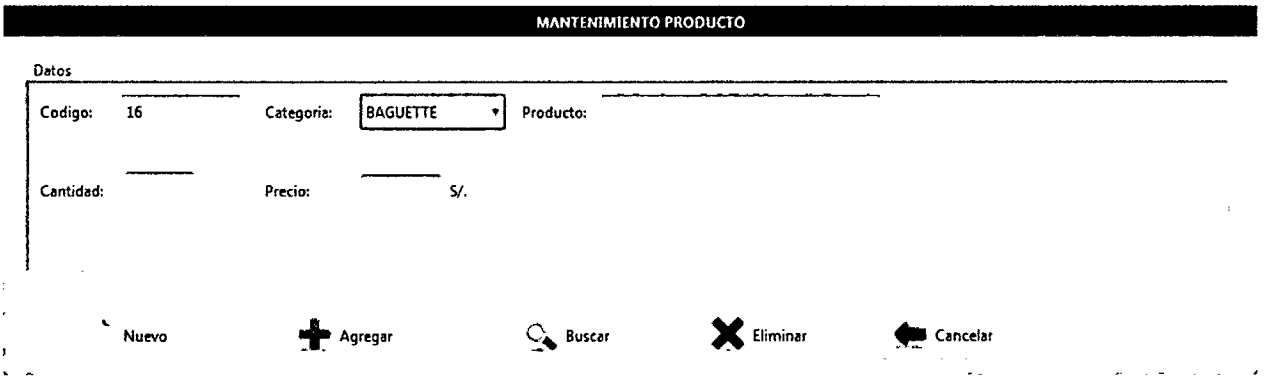


Figura 4.4.3.4 Mantenimiento de Producto

Mantenimiento Proveedor

Esta pestaña se encargará de ingresar, actualizar y eliminar y filtar por categorías los proveedores de la panadería para así tener un control de ellos además de una administración de la información rápida y eficaz para cuando se necesite.

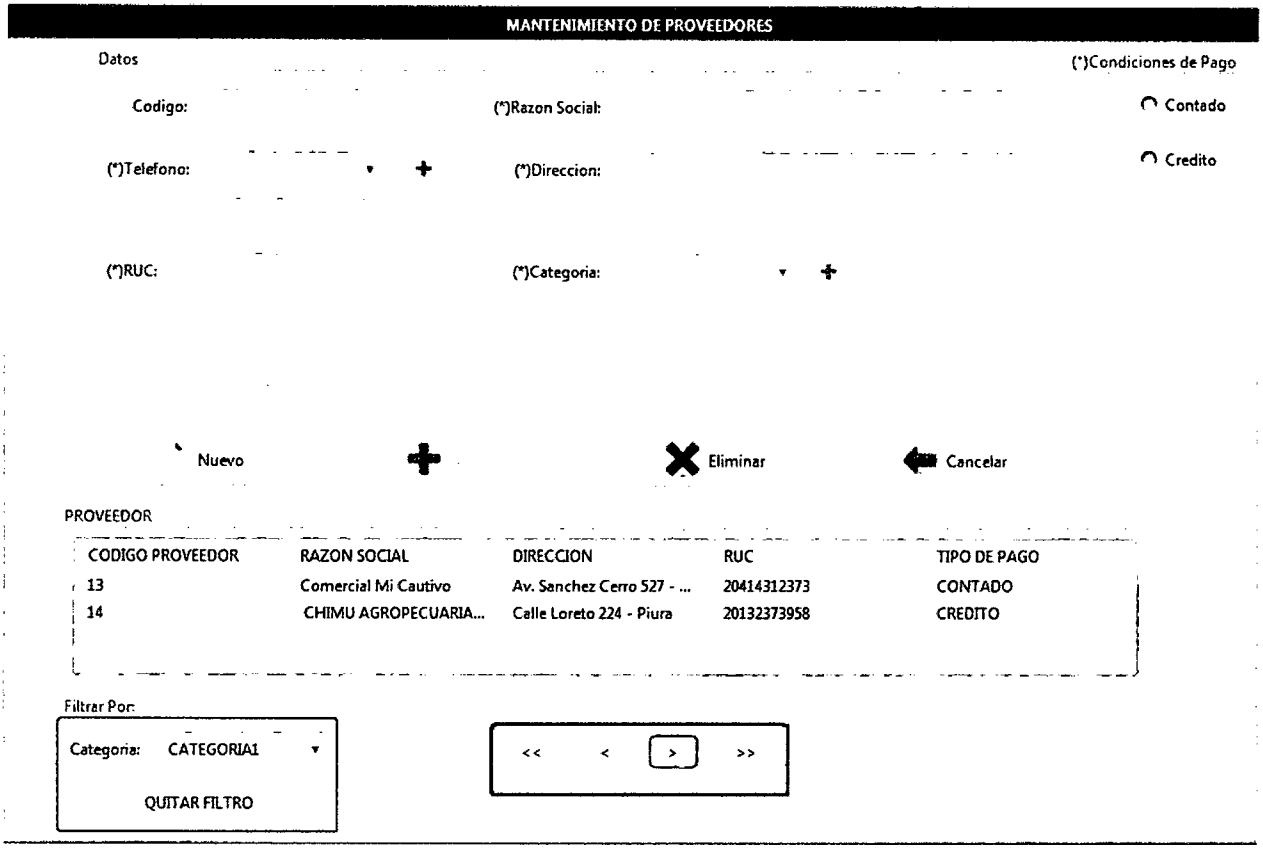


Figura 4.4.3.5 Mantenimiento de proveedor

Mantenimiento de Usuarios

El administrador registra a todos los empleados que tendrán acceso al sistema según el acceso que se necesite.

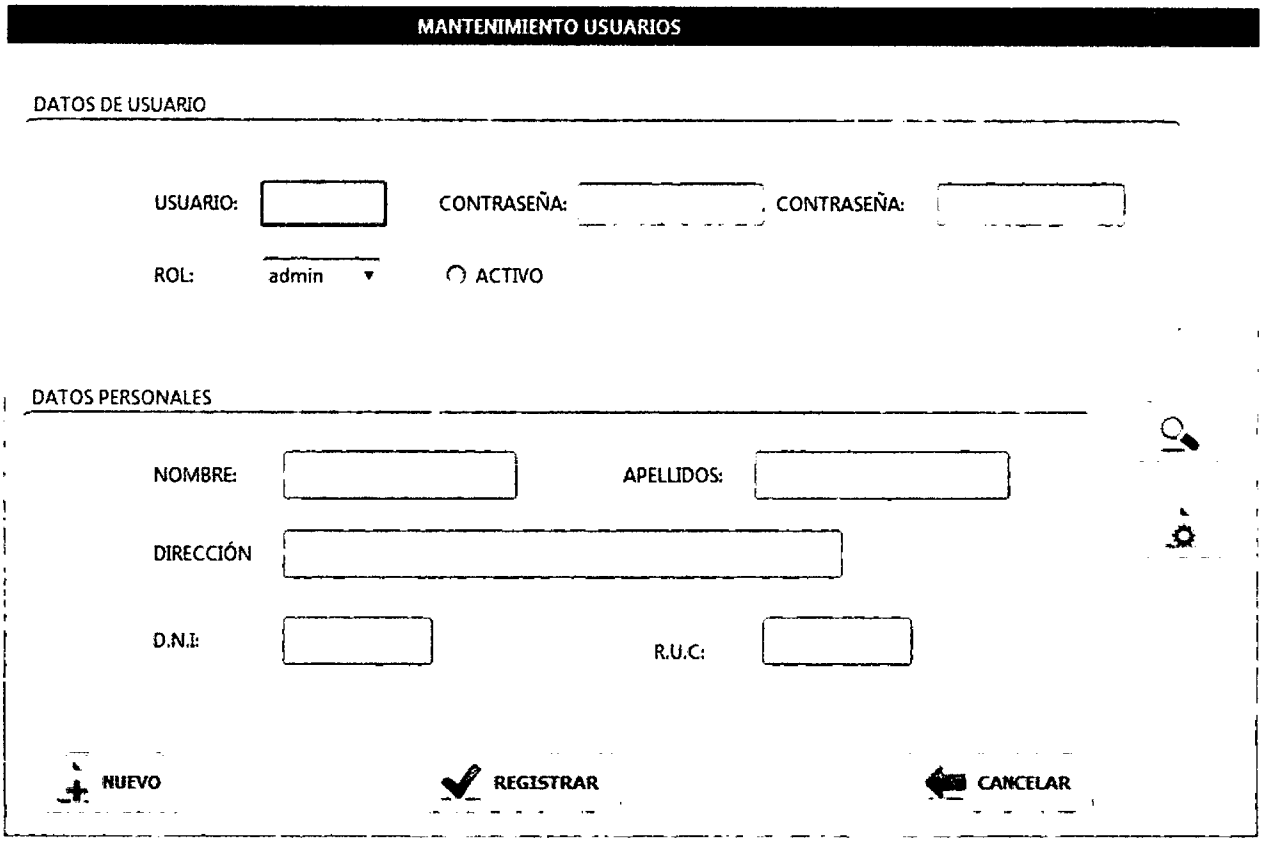


Figura 4.4.3.6 Mantenimiento de usuarios

Los datos pesonales de los Empleados serán cargados automáticamente despues de buscarlos en la ventana BUSCAR de la ventana principal.

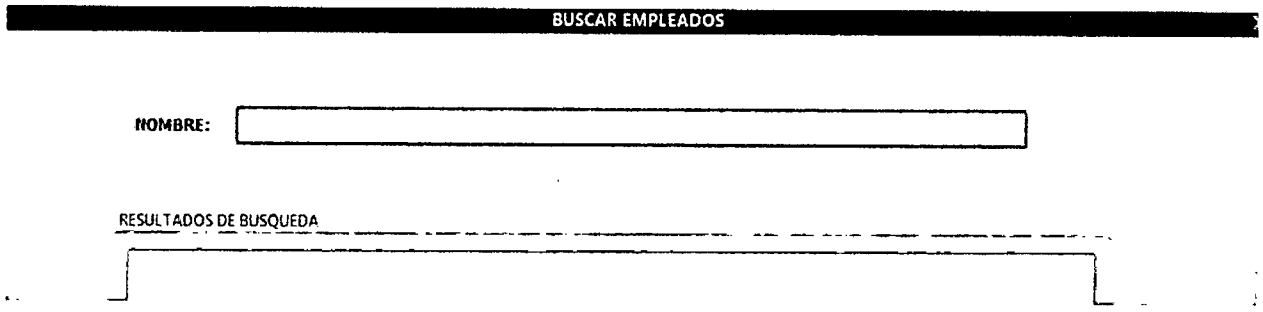
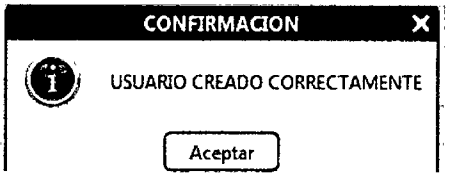


Figura 4.4.3.7 Buscar empleados

Al ingresar el usuario nos muestra el mensaje de confirmacion



Además para registrar los empleados para su próxima creación de Usuario en una ventana procedente del segundo boton lateral derecho.

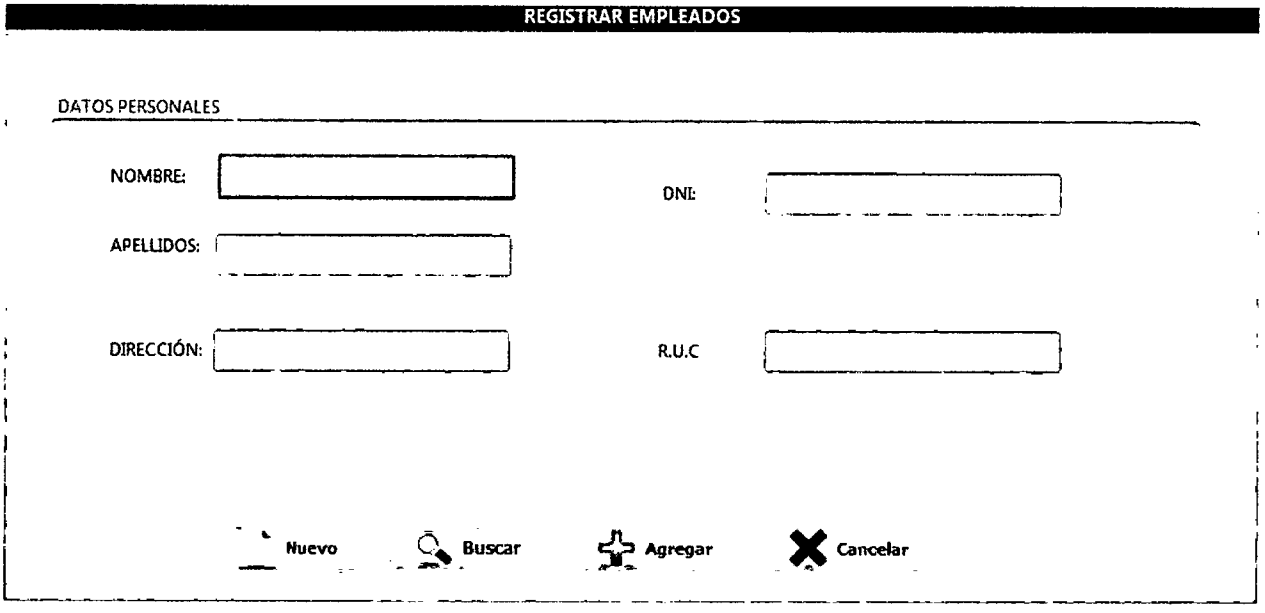
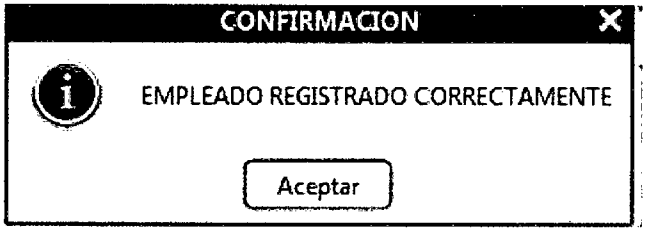


Figura 4.4.3.8 Registrar empleados

Se registra el empleado y arroja un mensaje de confirmacion para afirmar el registro.



Listar Productos

Para listar los productos según sus categorías y todos, tenemos esta ventana. Que permitirá revisar la gama de productos finales que fabrica la empresa.

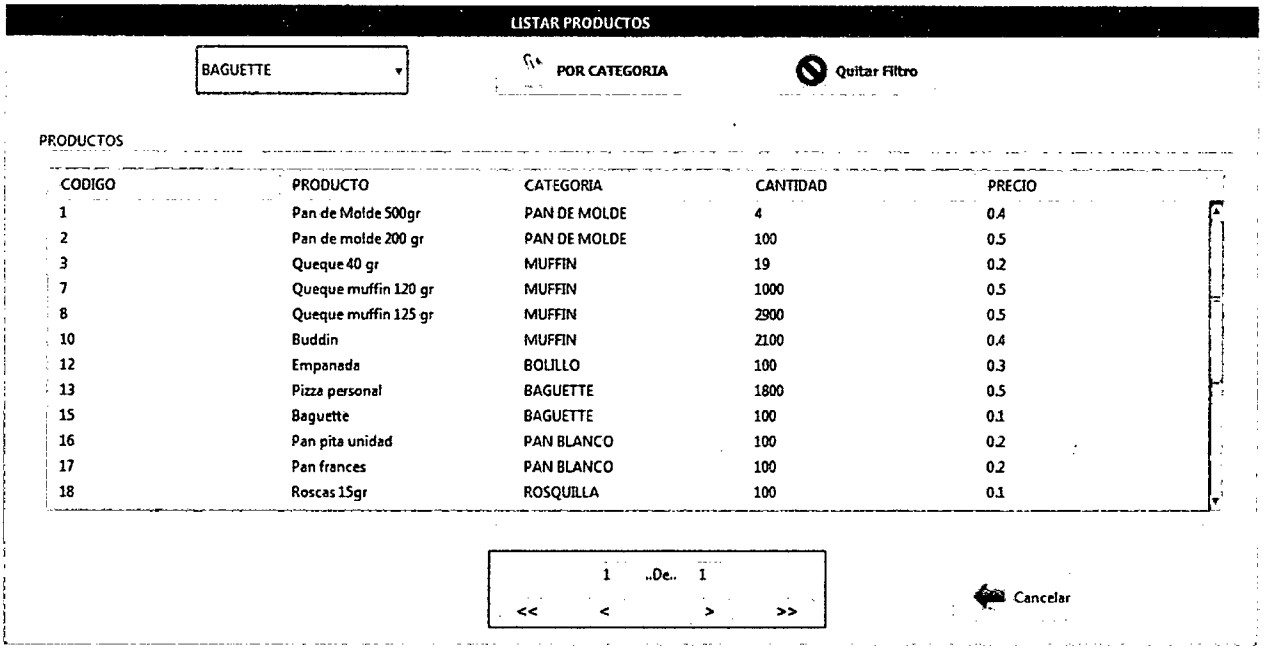


Figura 4.4.3.8 Listar Productos

4.4.4. Pedidos

En este módulo se llevará a cabo la administración de todos los pedidos de clientes que compren al por mayor y todos los clientes finales que hagan sus pedidos para fechas establecidas. Así se tendrá un control total de la entrega de aquellos pedidos sin problema.

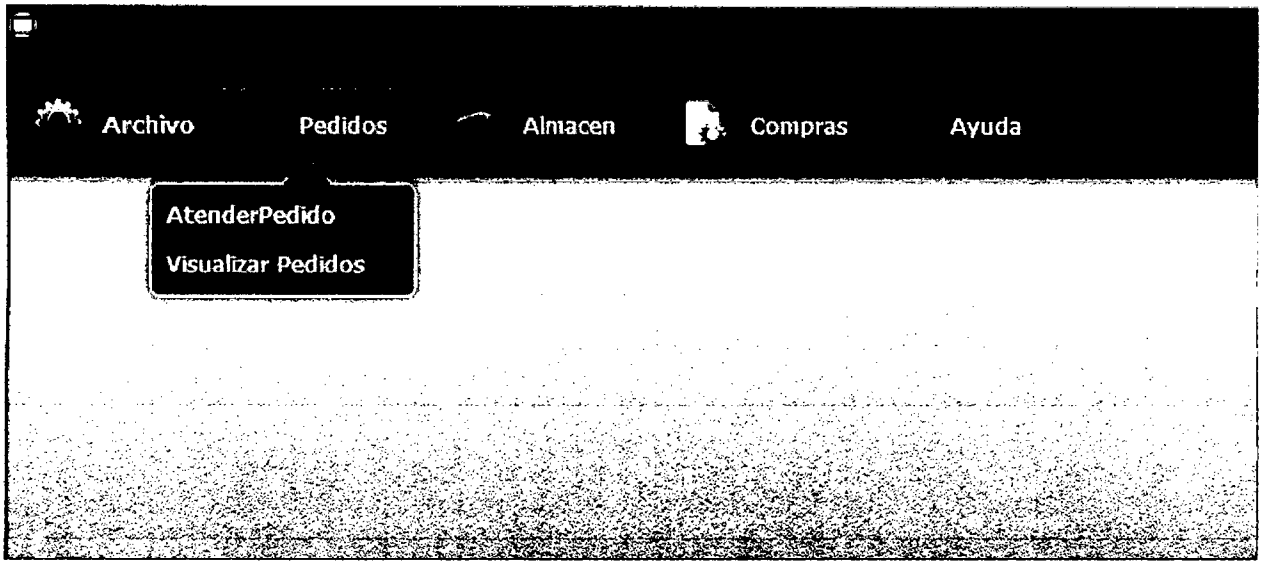


Figura 4.4.4.1 Registrar empleados

Para registrar el pedido se irá cargando los productos uno a uno según una búsqueda sensible.

pedis

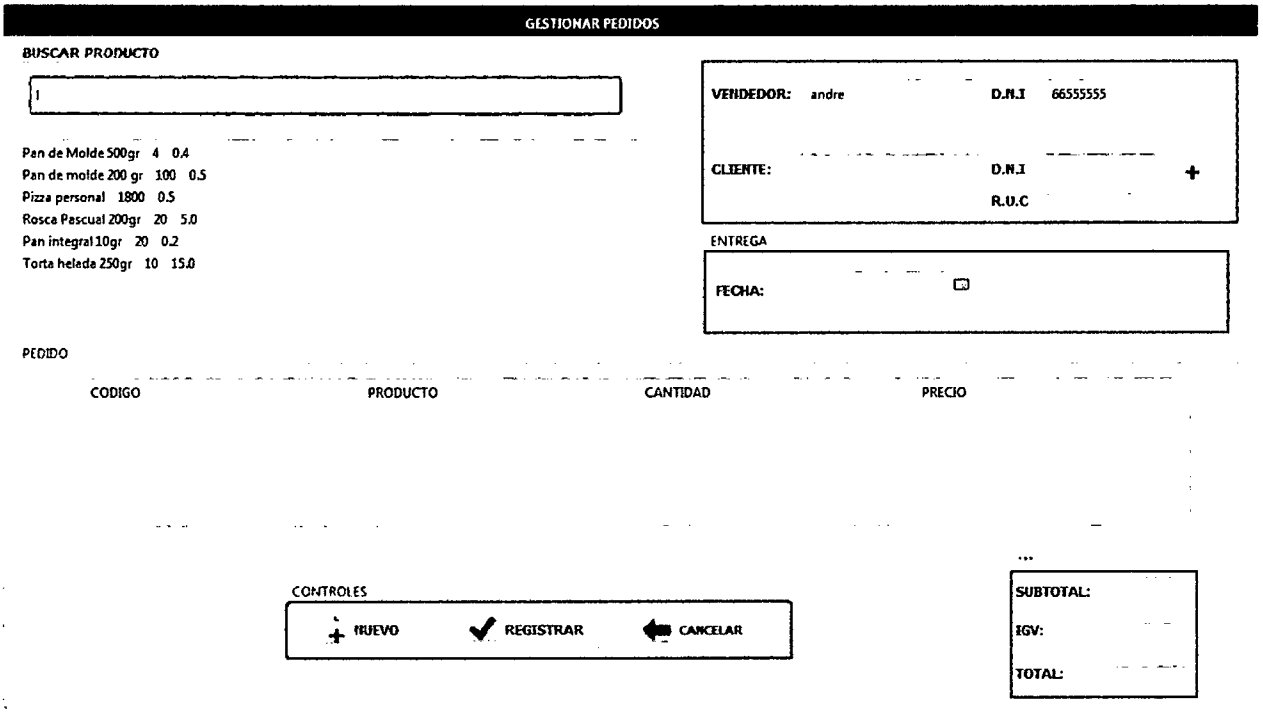


Figura 4.4.4.2 Gestionar pedido

Para ingresar los datos del cliente se cargarán automáticamente de una búsqueda sensible ya

antes mostrada.

Visualizar Pedidos

Para visualizar los pedidos, por fechas, los más recientes o los que ya están atendidos y los que aún no lo están se tiene esta pestaña que va a permitir las funciones ya mencionadas.

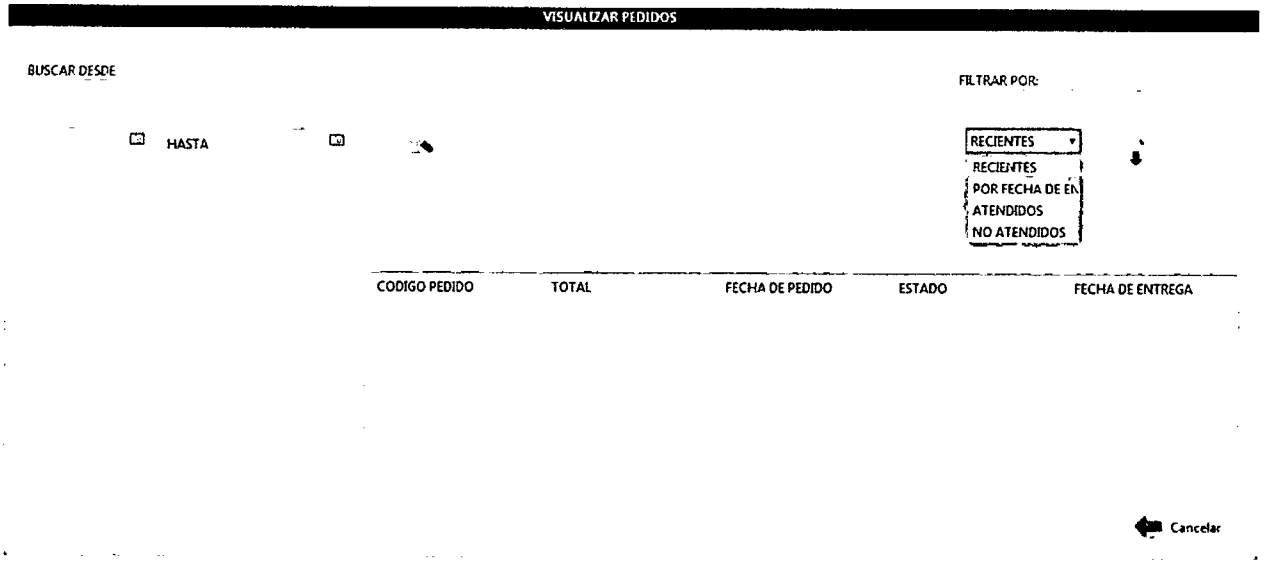


Figura 4.4.4.3. Visualizar Pedidos

Además los pedidos se filtrarán por los atendidos y los no atendidos, de los pedidos no atendidos se podrá elegir los que se vayan atendiendo y cambiarles es estado para que ya no aparezcan como pendientes.

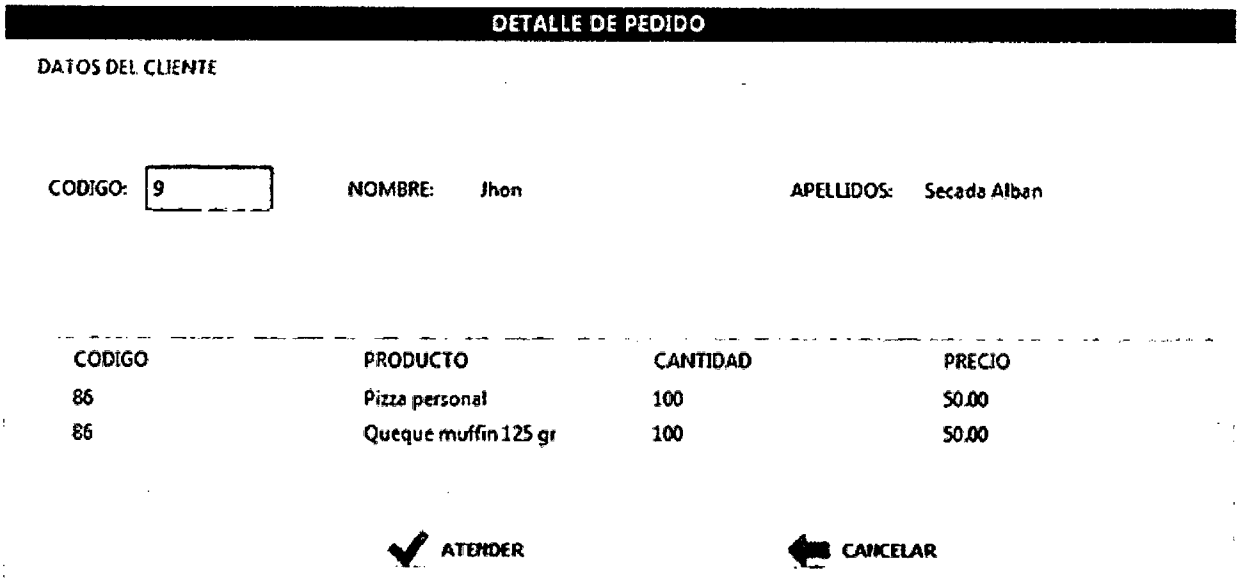
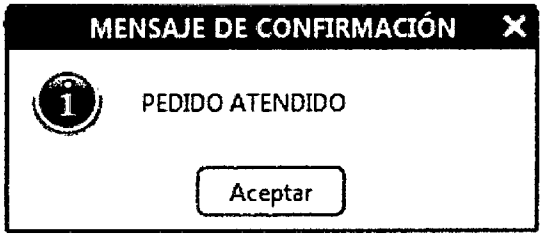
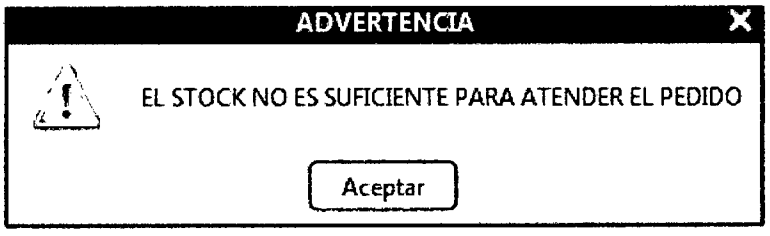


Figura 4.4.4.4. Detalle Pedidos

Al momento de atender el pedido saldrá un mensaje de confirmación que asegurará la acción.



Si no se cuenta con el stock suficiente del producto para atender el pedido el sistema arrojará un mensaje de alerta el cual no permitirá atender el pedido en el sistema.



Si el pedido no es pedido en su fecha pactada, al día siguiente de esta pasará a ser ANULADO y eliminado automáticamente de la data.

4.4.5. Almacén

Este módulo administrará la información de todos los insumos que se encuentren en almacén, así como las salidas de los mismos.

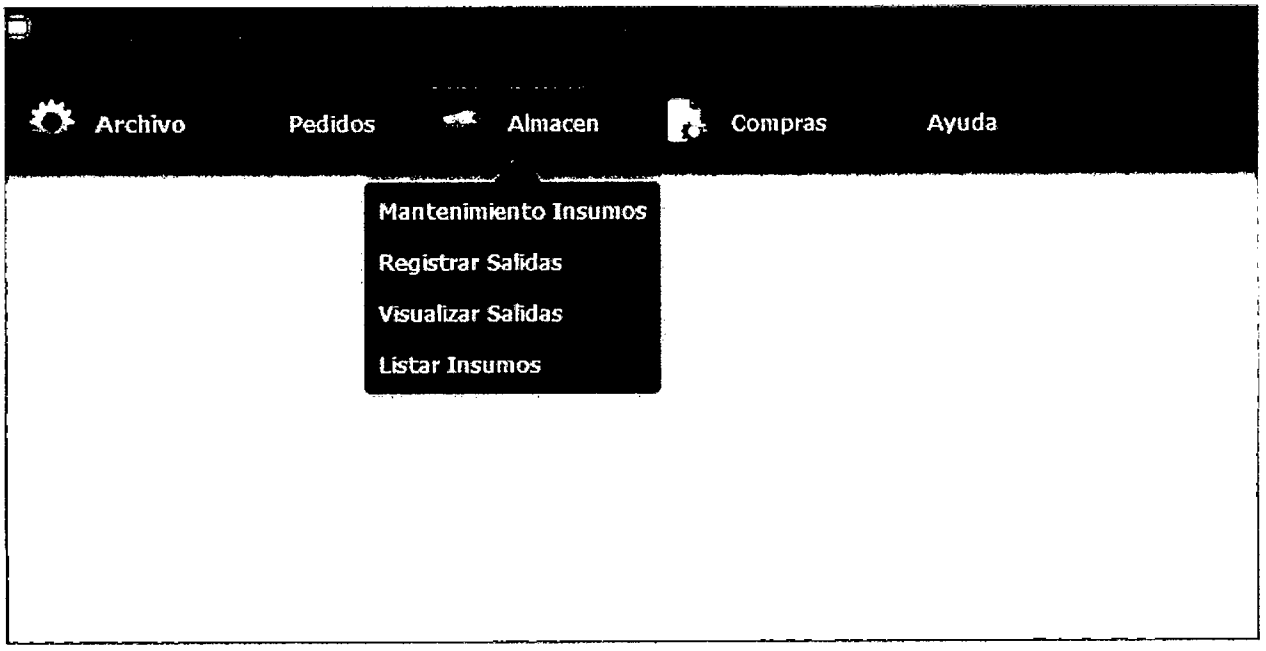


Figura 4.4.5.1 Almacén

Mantenimiento Insumo

Para registrar todos los insumos con sus respectivos datos tendremos esta ventana la cual almacenará además la cantidad mínima de cada uno de los insumos para que al momento que estos se encuentren por debajo de esta solicite la compra inmediata.

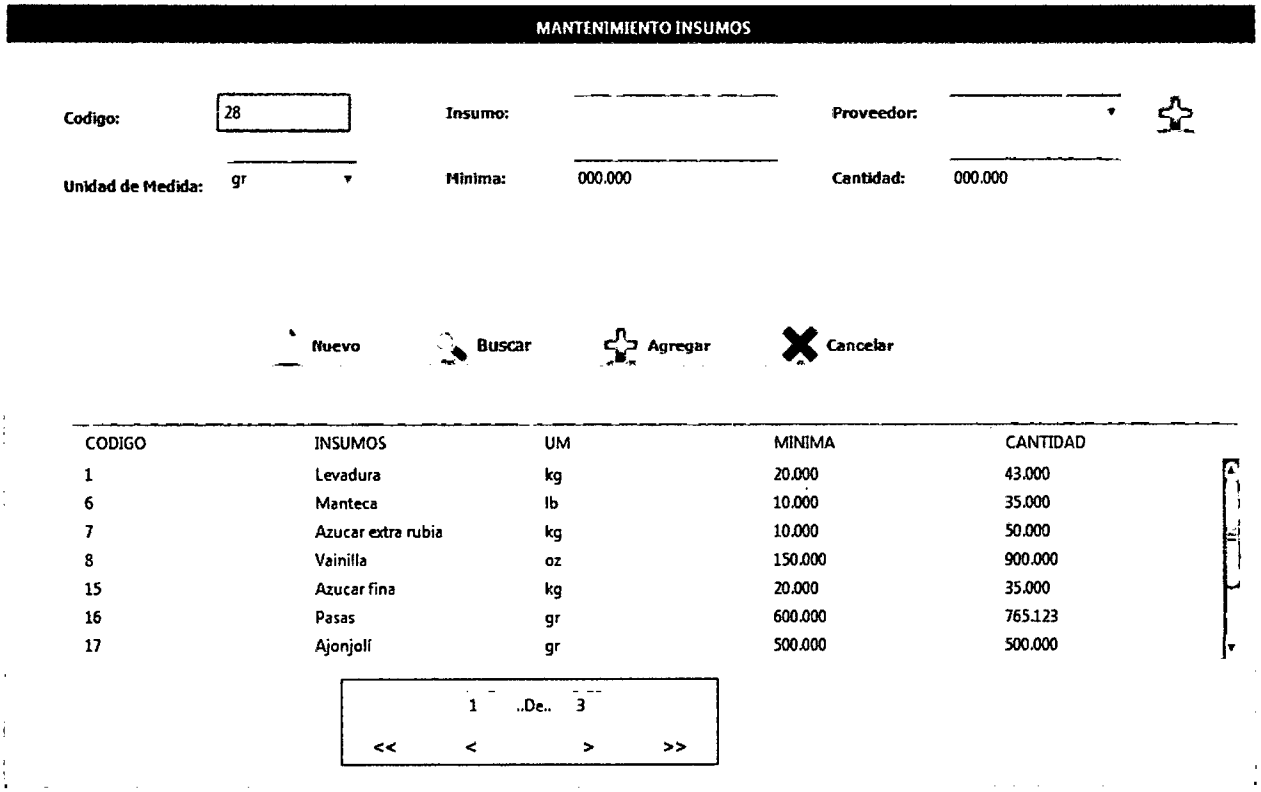
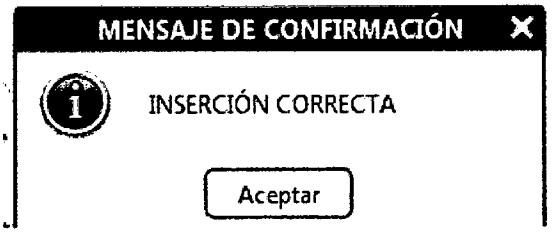


Figura 4.4.5.2 Mantenimiento de Insumos

Los proveedores se cargarán al comboBox que se presenta en la ventana al buscarlos mediante la ventana de búsqueda.

Al momento de insertar un nuevo registro el sistema arrocjará un mensaje de confirmación para validar el ignreso.



El boton BUSCAR también llamará a la ventana de búsqueda que hará la consulta por el nombre

de cada uno de los insumos para ver su estado actual.

Registrar Salidas

Registrará según la producción la salida de materia prima del almacén, este disminuirá en el stock actual y permitirá saber la situación de cada uno de los insumos al momento de generar las órdenes de compra. Además los insumos se podrán calcular según los productos requeridos en el día.

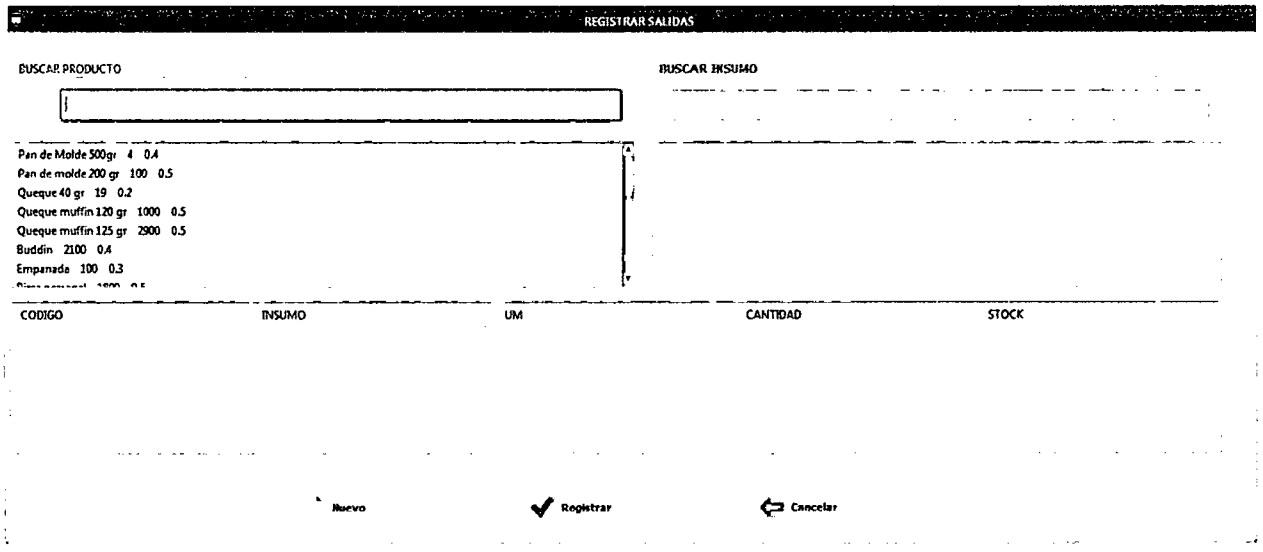


Figura 4.4.5.3 Registrar Salidas

Visualizar salida

En esta pestaña podremos viasualizar todas las salidas realizadas de insumos y el empleado que las realizó.

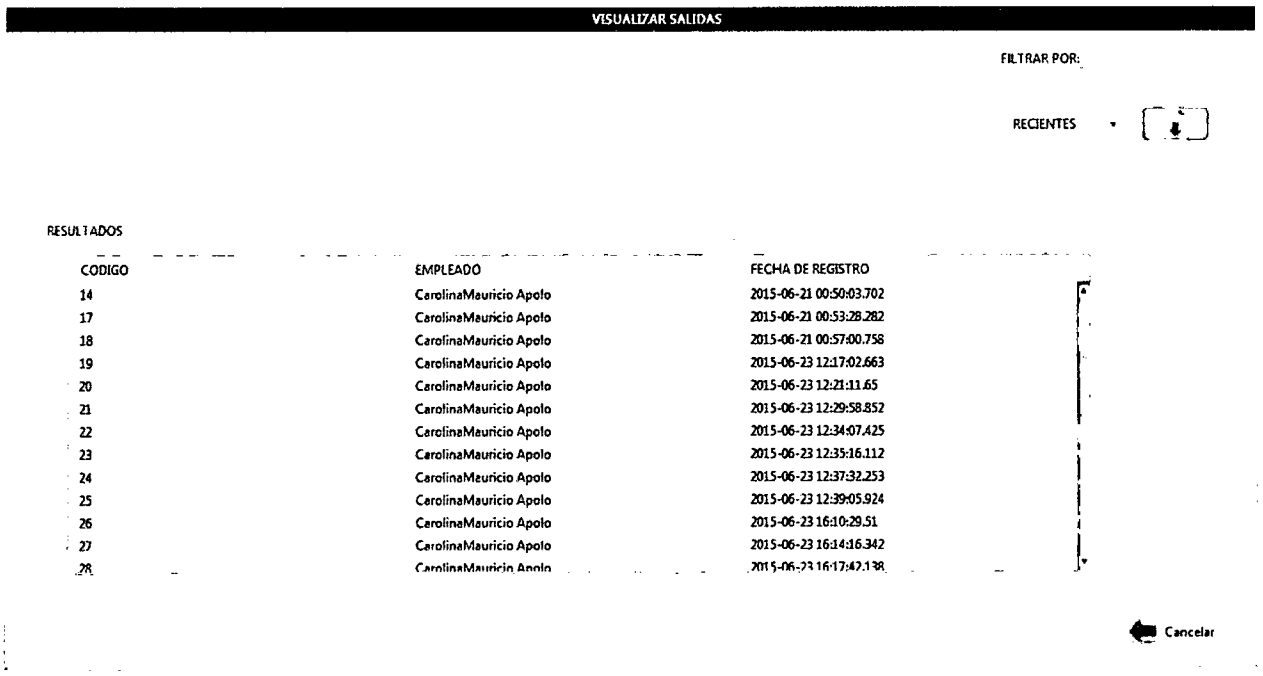


Figura 4.4.5.4 Visualizar Salida

Al seleccionar alguna de estas salidas se podrá visualizar su detalle, el insumo y sus datos.

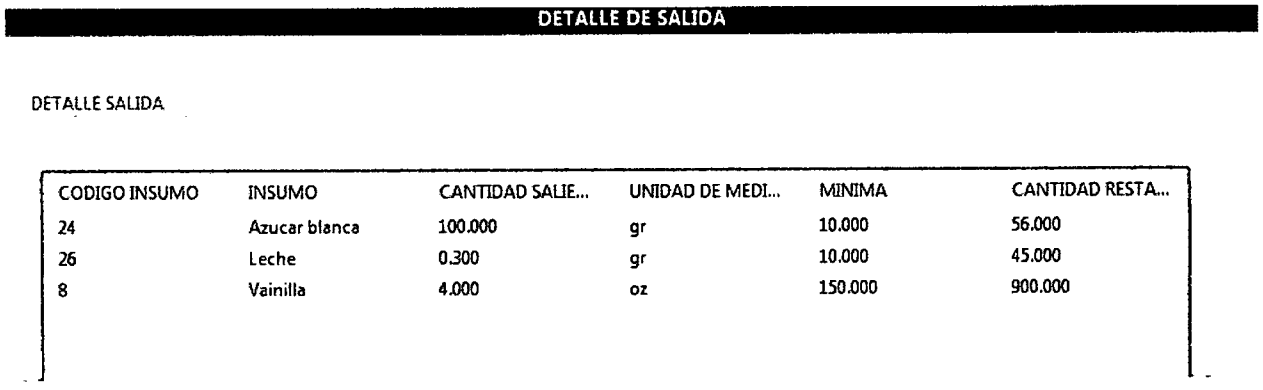


Figura 4.4.5.4 Detalle de Salida

Listar Insumos

Lista el total de insumos que hay actualmente en almacen para el inventario según las programaciones del administrador.

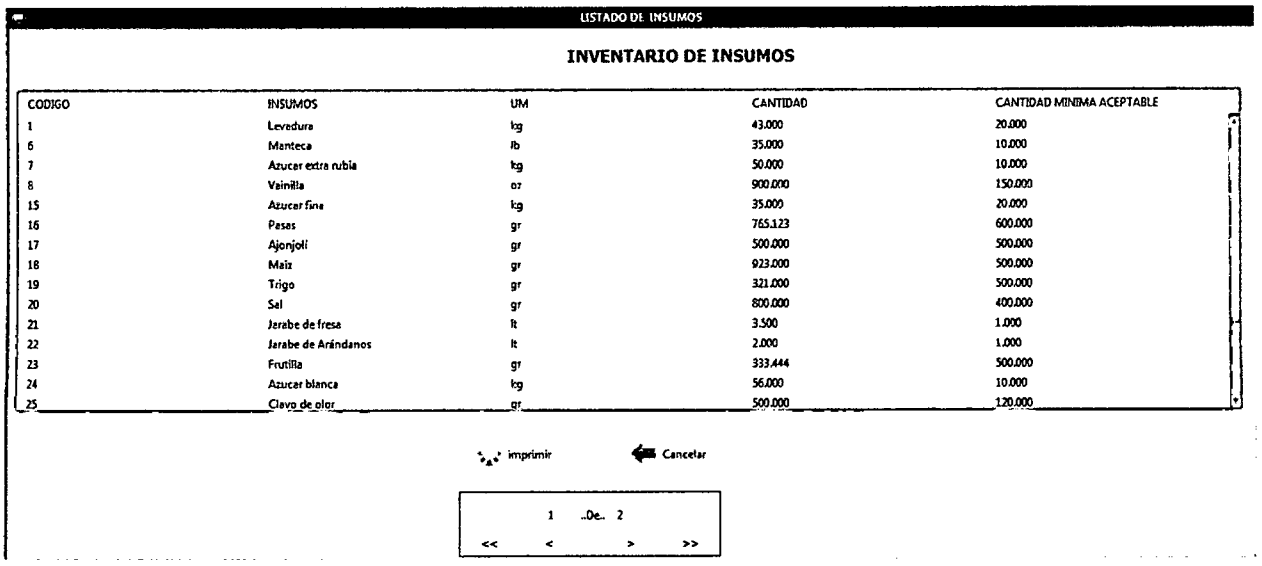


Figura 4.4.5.5 listar Insumos

4.4.6 Compras

El módulo de compras se encargará de listar los insumos que estén por debajo de su cantidad mínima y emitirá la Orden de compra para hacerla llegar al proveedor o proveedores correspondientes. Al mismo tiempo sugerirá una cantidad promedio para el pedido de insumos y que no haya un sobrestock en almacen.

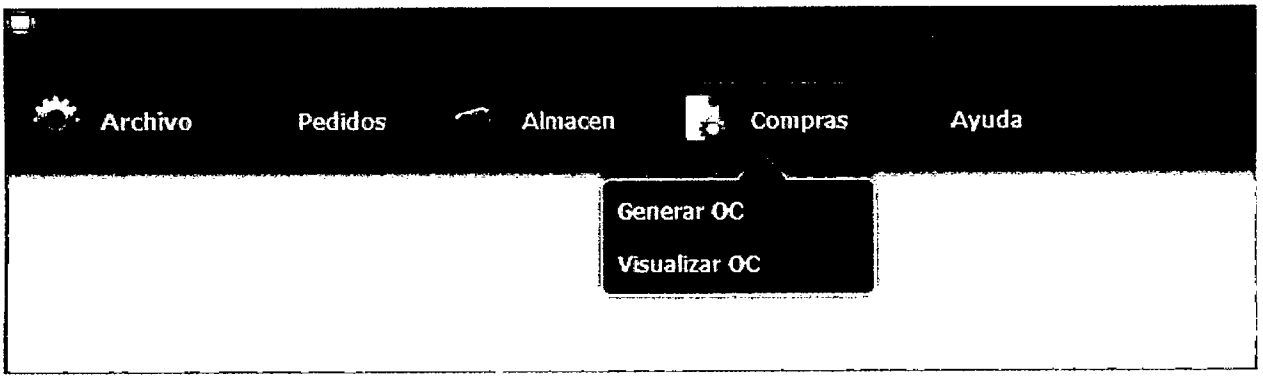


Figura 4.4.6.1 Compras

Listará los pedidos que estén por debajo de su cantidad mínima y se podrá incluir en la orden de compra para su adquisición inmediata.

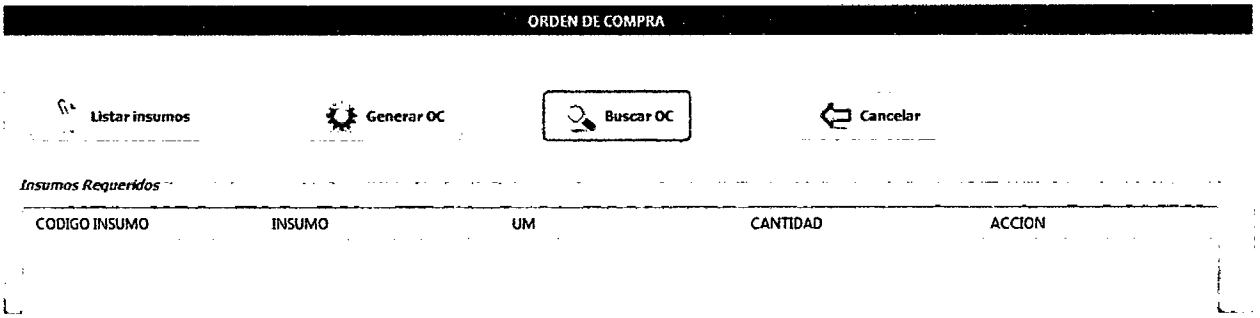
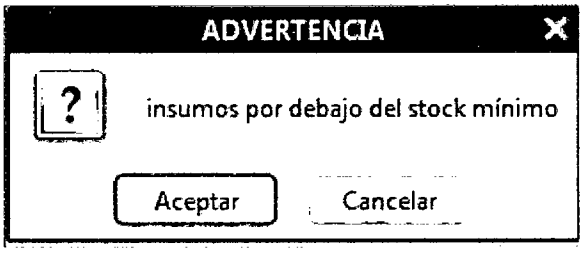


Figura 4.4.6.2 Orden de compra

Si hay productos que está por debajo de su cantidad mínima el usuario recibirá una alerta al momento ejecutar el sistema.



Al aceptar llevará a la ventana para generar la OC, si no se acepta, recordará la alarma cada 30 minutos.

Visualizar Orden de compra

Buscará las órdenes de compra según un rango de fechas seleccionadas por el usuario y nos mostrará cada una de las órdenes de compra y su detalle.

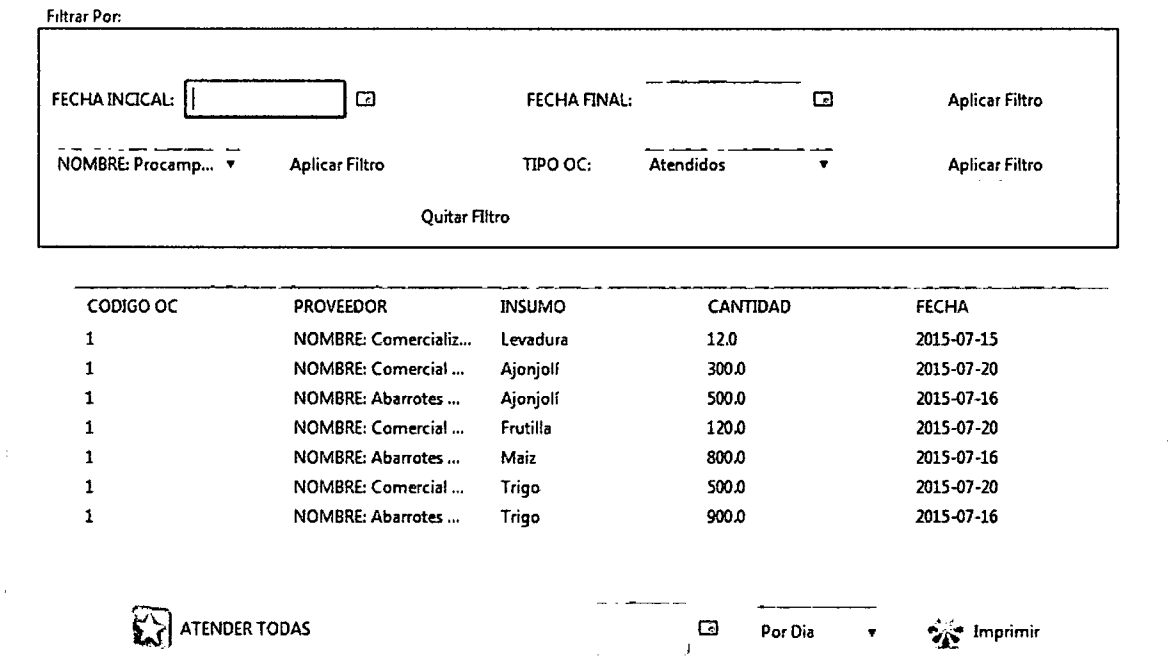


Figura 4.4.6.3 Orden de compra

Para atender la orden de compra se contará con la siguiente ventana:

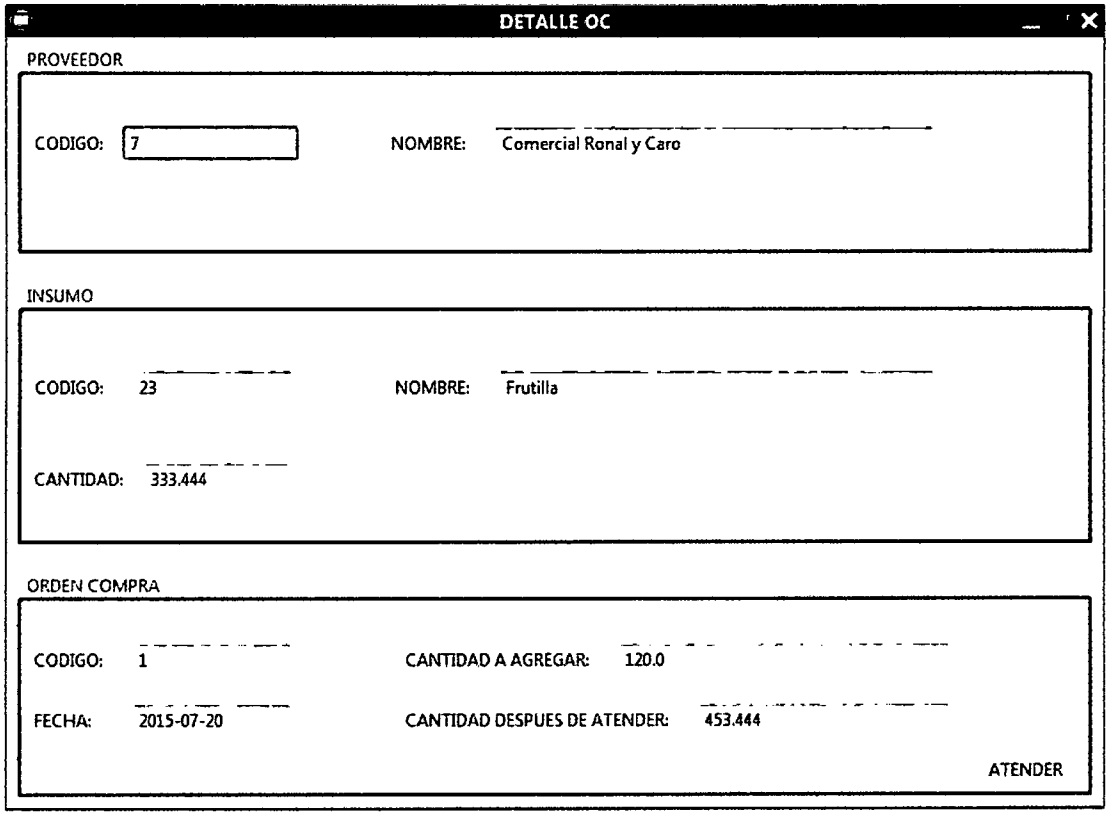


Figura 4.4.6.4 Detalle Orden de compra

CAPÍTULO 5. IMPLEMENTACION Y EVALUACIÓN DEL SISTEMA

5.1 DESARROLLO DEL SISTEMA

El desarrollo del sistema presentado se ha desarrollado en el lenguaje de Programación de código libre JAVA utilizando el entorno de desarrollo NETBEANS que también es un producto libre y sin restricciones de uso.

5.1.1. DTO:

El sistema de gestión de logística para panadería ha sido elaborado con programación orientada a objetos. Usando como patrones objetos DTO encargados de almacenar los datos de cada uno de ellos y facilitarlos a la base de datos.

A continuación la lista de clases utilizadas en el sistema y sus fuoncionalidades:

- **INSUMOSDTO**

```
public class InsumosDTO {
    private Integer idInsumo;
    private String insumo;
    private String  um;
    private BigDecimal minima;
    private BigDecimal cantidad;
    private BigDecimal salidas;
    private ArrayList<ProveedorDTO> arProv;
```

Aquí se pueden ver todos los atributos que tendrá el objeto INSUMO y que se mostrarán en la ventana respectiva.

- **UNIDADMEDIDADTO**

```
public class UnidadMedidaDTO {
private Integer idUnidad;
private String unidad;
private String ab;
```

Esta clase permitirá asignar la unidad de medida para que estas queden almacenadas y el objeto insumo elija una de estas.

- **SALIDASDTO**

```
public class SalidasDTO {
    private Integer idSalida;
    private Integer idEmpleado;
    private Integer idInsumo;
    private Timestamp fecha;
    private BigDecimal cantidad;
    private String um;
    private String nomComp;
    private Integer Idpersona;
```

La clase salidas será la encargada de recoger los datos generales de las salidas de insumos del almacén.

- **DETALLESALIDASDTO**

```
public class DetalleSalidasDTO {

    private Integer Idsalida;
    private ArrayList<InsumosDTO> arProducto;
```

Ya que la clase pasada tiene como atributos los datos generales de las salidas. Esta clase se encargará de cargar el Arreglo que contendrá la lista de insumos a almacenarse para su salida y la actualización del stock.

- **ORDENCOMPRA**

```
public class OrdenCompra {
    private int idOrdenCompra;
    private List<InsumosDTO> insumos;
    private double cantidad;
    private Date fechaOC;
    private ProveedorDTO proveedor;
```

En el caso de órdenes de compra estas estarán almacenadas en la siguiente clase que llama a la lista de insumos los cuales serán requeridos y que están en su cantidad mínima.

- **PEDIDODTO**

```
public class PedidoDTO {
   Integer idPedido;
    Integer idEmpleado;
    Integer idCliente;
    BigDecimal total;
    String fecha;
    boolean estado;
    Timestamp fechaEntrega;
```

Al igual que la clase salida, tendremos una clase pedido que nos permitirá guardar los datos generales de los pedidos, además del empleado que lo está registrando.

- **DETALLEPEDIDODTO**

```
public class DetallePedidoDTO {
    private Integer idPedido;
    private ArrayList<ProductoDTO> arProducto;
```

La clase detalle pedido se encargará de contar como atributo el arreglo de productos requeridos en cada uno de los pedidos y asociarse a los datos generales presentados en la clase anterior.

- **PRODUCTODTO**

```
public class ProductoDTO {

    private Integer idCategoria;
    private String producto;
    private Integer cantidad;
    private BigDecimal precio;
    private Integer idProducto;
```

La clase producto almacena datos de la producción diaria, estos serán ingresados en el arreglo de detalle de pedido para su atención.

- **USUARIODTO**

```
public class UsuarioDTO {

    private Integer id;
    private String contrasenha;
    private  String usuario;
    private boolean estado;
    private String nombre;
    private boolean idRol;
```

La clase usuario permite guardar los datos de cada uno de los uduarios que se ingresen al sistema

según su tipo o rol que se presenta en la siguiente clase.

- **ROLDTO**

```
public class RolDTO {
    private boolean idRol;
    private String rol;
```

Se presentarán dos roles: ADMINISTRADOR y VENDEDOR, el primero tendrá acceso a todo el sistema y el segundo sólo a consultas por seguridad del sistema.

- **CATEGORIADTO y CATEGORIAPROVEEDOR**

```
public class CategoriaDTO {
    private Integer idCategoria;
    private String categoria;


public class CategoriaProveedor {
    private Integer idCategoria;
    private String categoria;
```

Ambas clases tienen solo dos atributos, la primera que almacenará las categorías de productos y la segunda de los proveedores para su búsqueda rápida.

- **TELÉFONODTO**

```
public class TelefonoDTO {

    private Integer idTelefono;
    private Integer idPersona;
    private String numTelefono;
```

Contiene atributos del objeto para el número telefónico, ya que estos pueden ser varios para una sola persona y se necesitará registrarlos todos.

- **HERENCIA**

Para el desarrollo de las clases Cliente, Proveedores y empleado será creada una clase Padre o superclase que contenga los atributos en común de estas 3 clases y que le heredarán estos.

La superclase que contendrá estos atributos será la clase PERSONA la cual almacenará los datos generales de las clases antes mencionadas.

- **PERSONADTO**

```
public class PersonaDTO {

    private Integer idPersona;
    private String nombre;
    private String apellidos;
    private String direccion;
    private String dni;
    private String ruc;
    private List<TelefonoDTO> telefono;
```

Las clases hijas o subclases que heredarán los atributos y métodos de estas son las siguientes:

- **CLIENTEDTO**

```
public class ClienteDTO extends PersonaDTO {
    private Integer idCliente;

    public ClienteDTO()
    {
       super();
    }
```

- **PROVEEDORDTO**

```
public class ProveedorDTO extends PersonaDTO {

    private int idProveedor;
    private boolean tipoPago;
    private BigDecimal precio;

    public ProveedorDTO() {
       super();
    }
```

- **EMPLEADODTO**

```
public class EmpleadoDTO extends PersonaDTO{
    private Integer idEmpleado;
    private Integer idUsuario;
    private Integer idPersona;
```

De esta manera las clases hijas tendrán todas las características de la clase Padre PERSONA y permitirá un ahorro de recursos en líneas de código y acceso a memoria.

Nota: Todas las los objetos DTO tienen como métodos los getter and setter, el código presentado en el CD adjunto en anexos.

5.1.2 DAO:

Para encapsular el acceso a la base de datos y así poder crear una capa de persistencia que nos facilitará interactuar con esta de una manera independiente donde se llamarán las funciones que se hayan creado en el gestor de base de datos utilizado, en este caso Postgresql.

A continuación detallaremos algunas de estas clases así como sus métodos. Haciendo hincapié que el código completo se mostrará en el Cd entregado adjunto del presente informe.

- **INSUMODAO**

Al crear la clase se llamará a la conexión a la BD y se capturarán las excepciones que puedan desarrollarse en el momento esta.

```
public class InsumosDAO {

    Connection con=null;
    PreparedStatement pst=null;
    ResultSet rs= null;

    public InsumosDAO()
    {
        try
        {
           con=ConexionJDBC.Conectar();
        }catch(SQLException e)
        {
            e.printStackTrace();
        }

    }
    public void registerEmployes(){

        try{
            pst = con.prepareStatement(null);

        }catch(SQLException e){

        }
```

Cada vez que se llame a una función de la BD y que se termine la operación se tendrá que cerrar la conexión a esta para una mayor eficiencia.

```
private void close()
{
    try
    {
        if(rs!=null && pst!=null)
        {
            pst.close();
            rs.close();
        }

    }catch(SQLException r)
    {
    }
}
```

El ID de cada uno de los insumos será un número correlativo que se irá asignando a medida que se ingresen estos y se guarden en la BD.

Todos los ID de los demás objetos se obtendrán de esta manera en sus respectivas clases DAO.

```
public Integer getNextIdInsumo()
{
    Integer id=null;
    try
    {
        pst=con.prepareStatement("select fnc_getnextidprimaryproduct()");
        rs=pst.executeQuery();
        while(rs.next())
        {
            id=rs.getInt(1)+1;
        }
    }catch(SQLException e)
    {
        e.printStackTrace();
    }
    finally
    {
        close();
    }
    return id;
}
```

Este método nos retornará el arreglo de atributos de insumos registrados en la BD para hacer el listado de estos.

```
public ArrayList<InsumosDTO> selectAllPrimaryProduct()
 {

    ArrayList<InsumosDTO> arIn= new ArrayList<InsumosDTO>();
    ArrayList<ProveedorDTO> prov;
     try
     {
        pst=con.prepareStatement("select * from fnc_selectallprimaryproducts();");
        rs=pst.executeQuery();
        while(rs.next())
        {
            if(!arIn.isEmpty())
            {
                int i=0;
                boolean flag= false;
                while(i<arIn.size())
                {

                    if(arIn.get(i).getIdInsumo().equals(rs.getInt(1)))
                    {
                        ProveedorDTO pdto = new ProveedorDTO();
                        pdto.setIdProveedor(rs.getInt(7));
                        pdto.setNombre(rs.getString(8));
                        pdto.setDireccion(rs.getString(10));
                        pdto.setDni(rs.getString(11));
                        pdto.setRuc(rs.getString(12));
                        arIn.get(i).getArProv().add(pdto);
                        flag=false;
                        rs.next();
                        i=0;
                    }
                    else
                    {
                        flag=true;
                        i++;
                    }
                }
                if(flag)
                {
                        prov = new ArrayList<ProveedorDTO>();
                        ProveedorDTO pdto = new ProveedorDTO();
                        pdto.setIdProveedor(rs.getInt(7));
                        pdto.setNombre(rs.getString(8));
                        pdto.setDireccion(rs.getString(10));
                        pdto.setDni(rs.getString(11));
                        pdto.setRuc(rs.getString(12));
                        pdto.setPrecio(rs.getBigDecimal(6));

                    prov.add(pdto);
                    arIn.add(new InsumosDTO(rs.getInt(1), rs.getString(2), rs.getString(3),
                            rs.getBigDecimal(4),
```

```
                rs.getBigDecimal(5),prov));
            }
        }else
        {
            prov = new ArrayList<ProveedorDTO>();
            ProveedorDTO pdto = new ProveedorDTO();
            pdto.setIdProveedor(rs.getInt(7));
            pdto.setNombre(rs.getString(8));
            pdto.setDireccion(rs.getString(10));
            pdto.setDni(rs.getString(11));
            pdto.setRuc(rs.getString(12));
            pdto.setPrecio(rs.getBigDecimal(6));

            prov.add(pdto);
            arIn.add(new InsumosDTO(rs.getInt(1), rs.getString(2), rs.getString(3),
                rs.getBigDecimal(4),
            rs.getBigDecimal(5),prov));
        }
```

La búsqueda se hará de una manera sensitiva según vaya ingresando el nombre del insumo, primero cargaremos todos los insumos en un arreglo en la memoria, la búsqueda se realizará desde la memoria, así evitaremos innecesarios ingresos a la BD y un ahorro del recurso tiempo.

```
public ArrayList<InsumosDTO> sensitiveSearchInsumo()
{
    ArrayList<InsumosDTO> arInDTO= new ArrayList<InsumosDTO>();
    try
    {
        pst=con.prepareStatement("select * from insumos;");
        rs=pst.executeQuery();
        while(rs.next())
        {
            arInDTO.add(new InsumosDTO(rs.getInt(1), rs.getString(2), rs.getString(3),
                BigDecimal.valueOf(rs.getDouble(4)), rs.getBigDecimal(5)));
        }
    }catch(SQLException e)
    {
    }
    return arInDTO;
}
```

Llamamos a actualizar insumos con el siguiente método capturando los datos ingresados en la ventana y almacenándolos en la BD

```
public void updatePrimaryProduct(InsumosDTO idto)
{
    try
    {
        pst= con.prepareStatement("select * from fnc_updateprimaryproduct("+idto.getIdInsumo()+
             ", '"+idto.getInsumo()+"','"+idto.getUm()+"',"+idto.getMinima()+","+idto.getCantidad()+");");
        rs=pst.executeQuery();
        while(rs.next())
        {
            JOptionPane.showMessageDialog(null, rs.getString(1));
        }
    }catch(SQLException e)
    {
        JOptionPane.showMessageDialog(null, "ERROR DE ACTUALIZACION", "ADVERTENCIA", JOptionPane.WARNING_MESSAGE);
    }
}
```

Para listar los productos que estén por debajo de su cantidad ínim tenemos el siguiente método que llamará a la función fnc_list_all_inputs() que será detallada más adelante.

```
public ArrayList<InsumosDTO> getAllInputsMinimum(){
    ArrayList<InsumosDTO> listaInputs = null;
    try {
        listaInputs = new ArrayList<>();
        pst = con.prepareStatement("select * from fnc_list_all_inputs();");
        rs = pst.executeQuery();
        while (rs.next()) {
            InsumosDTO input = new InsumosDTO();
            input.setIdInsumo(rs.getInt(1));
            input.setInsumo(rs.getString(2));
            input.setUm(rs.getString(3));
          listaInputs.add(input);

        }

        return listaInputs;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } finally {
        close();
    }
}
```

- **SALIDASDAO**

La conexión a la BD será la misma en todas las clases DAO. Para las demás funciones se presentarán los métodos a continuación:

En el siguiente método se capturarán todos los datos generales ingresados a la ventana para su registro, este devolverá un arreglo con los datos adquiridos.

```
public ArrayList<SalidasDTO> getOutputs()
{
    ArrayList<SalidasDTO> arSalDTO = new ArrayList<>();
    try
    {
        pst= con.prepareStatement("select * from fnc_filteroutputs();");
        rs=pst.executeQuery();
        while(rs.next()){
            arSalDTO.add(new SalidasDTO(rs.getInt(1), rs.getInt(2), rs.getTimestamp(5),
                    rs.getString(4), rs.getInt(3)));
        }
    }catch(SQLException e)
    {
        e.printStackTrace();
    }
     finally
    {
        close();
    }
    return arSalDTO;
}
```

Para registrar la salida en la BD agregará todos los datos generales, además del usuario que esté haciendo la salida de insumos del almacén, para llevar un control riguroso de quien realiza la disminución de insumos del almacén.

```
public Integer RegisterOutput(SalidasDTO sdto)
{
    Integer id=null;
    try
    {
        pst=con.prepareStatement("select * from "
                + "fnc_registeroutput("+sdto.getIdEmpleado()+");");
        rs=pst.executeQuery();
        while(rs.next())
        {
            id=rs.getInt(1);
        }
        if(id != null || id!=0 || id != -1){
            JOptionPane.showMessageDialog(null, "INSERCION EXITOSA");
        }else{
            JOptionPane.showMessageDialog(null, "FALLO LA INSERCION");
        }
    }catch(SQLException e)
    {
        JOptionPane.showMessageDialog(null, "FALLO LA INSERCION");
    }
    finally
    {
        close();
    }
    return id;
```

Para obtener el listado de salidas según el ingreso de ellas:

```
public ArrayList<SalidasDTO> getOutputsdesc() {
    ArrayList<SalidasDTO> arSalDTO = new ArrayList<>();
    try
    {
        pst= con.prepareStatement("select * from fnc_filteroutputsdesc();");
        rs=pst.executeQuery();
        while(rs.next()){
              arSalDTO.add(new SalidasDTO(rs.getInt(1), rs.getInt(2),
                    rs.getTimestamp(5), rs.getString(4), rs.getInt(3)));
        }
    }catch(SQLException e)
    {

    }
     finally
    {
        close();
    }
    return arSalDTO;
}
```

- **DETALLESALIDADAO**

La lista de productos de cada salida estará trabajada en la clase DetalleSalida y serán capturados en un arreglo de insumos.

```
public DetalleSalidasDTO getproductobySalidas(DetalleSalidasDTO ped)
{
    ArrayList<InsumosDTO> arinsdto= new ArrayList<InsumosDTO>();
    ped.setArProducto(arinsdto);
    try
    {
        pst= con.prepareStatement("select * from fnc_show_view_details_outputsprimaryproduct("+ped.getIdsalida()+");");
        rs=pst.executeQuery();
        while(rs.next())
        {
              ped.getArProducto().add(new InsumosDTO(rs.getInt(2), rs.getString(3),
              rs.getString(4), rs.getBigDecimal(5), rs.getBigDecimal(6), rs.getBigDecimal(7)));
        }
    }catch(SQLException e)
    {
        e.printStackTrace();
    }
    finally
    {
        close();
    }
    return ped;
```

Y se registrarán con el siguiente método:

```
public void registerOutput(DetalleSalidasDTO dsdto)
{
    try
    {
        int i=0;
        while(i<dsdto.getArProducto().size()){
            pst= con.prepareStatement("select * from fnc_registeroutputprimaryproduct("+dsdto.getIdsalida()
                    +","+dsdto.getArProducto().get(i).getIdInsumo()+","+dsdto.getArProducto().get(i).getCantidad()
                    +",'"+dsdto.getArProducto().get(i).getUm()
                + "');");
        rs = pst.executeQuery();
        rs.next();

        i++;
        }
    }catch(SQLException e)
    {
        e.printStackTrace();
    }
```

- **ORDENCOMPRA**

La operación para guardar la OC será funcionalizada en el siguiente método:

```
public Integer save_orden_compra(OrdenCompra oc, ArrayList<Double>
        cantidades, List<ProveedorDTO> proveedores) throws SQLException{
    Integer result = 0;

    try {
        con.setAutoCommit(false);

        for (int i = 0; i < oc.getInsumos().size(); i++) {
            pst = con.prepareStatement("select * from fnc_register_orden_compra(?,?,?,?);");
            pst.setInt(1, oc.getIdOrdenCompra());
            pst.setInt(2, oc.getInsumos().get(i).getIdInsumo());
            pst.setInt(3, proveedores.get(i).getIdProveedor());
            pst.setDouble(4, cantidades.get(i));

            rs = pst.executeQuery();
            while (rs.next()) {
                result = rs.getInt(1);
            }

            con.commit();
        }

        return result;
```

Para listar el total de órdenes de compra tendremos la siguiente operación:

```
public List<OrdenCompra> get_all_oc(){
    List<OrdenCompra> result = null;
    List<InsumosDTO> insumos = null;
    OrdenCompra oc =new OrdenCompra();
    try {
        pst = con.prepareStatement("select * from fnc_list_all_oc();");
        rs = pst.executeQuery();
        insumos = new ArrayList<>();
        result = new ArrayList<>();
        while (rs.next()) {

            oc= new OrdenCompra();
            oc.setIdOrdenCompra(rs.getInt(1));
            ProveedorDTO prov = new ProveedorDTO();
            prov.setNombre(rs.getString(2));
            oc.setProveedor(prov);
            InsumosDTO ins = new InsumosDTO();
            ins.setInsumo(rs.getString(3));
            insumos.add(ins);
            oc.setCantidad(rs.getDouble(4));
            oc.setFechaOC(rs.getDate(5));
            oc.setInsumos(insumos);
            result.add(oc);
        }

        return result;
```

```
public List<OrdenCompra> get_all_oc(){
    List<OrdenCompra> result = null;
    List<InsumosDTO> insumos = null;
    OrdenCompra oc =new OrdenCompra();
    try {
        pst = con.prepareStatement("select * from fnc_list_all_oc();");
        rs = pst.executeQuery();
        insumos = new ArrayList<>();
        result = new ArrayList<>();
        while (rs.next()) {

            oc= new OrdenCompra();
            oc.setIdOrdenCompra(rs.getInt(1));
            ProveedorDTO prov = new ProveedorDTO();
            prov.setNombre(rs.getString(2));
            oc.setProveedor(prov);
            InsumosDTO ins = new InsumosDTO();
            ins.setInsumo(rs.getString(3));
            insumos.add(ins);
            oc.setCantidad(rs.getDouble(4));
            oc.setFechaOC(rs.getDate(5));
            oc.setInsumos(insumos);
            result.add(oc);
        }

        return result;
```

De igual manera listaremos por proveedor:

```
public List<OrdenCompra> get_all_oc_by_proveedor(ProveedorDTO proveedor){
  List<OrdenCompra> result = null;
  List<InsumosDTO> insumos = null;
  OrdenCompra oc =new OrdenCompra();
 try {
     pst = con.prepareStatement("select * from get_oc_by_proveedor(?);");
     pst.setInt(1, proveedor.getIdProveedor());

     rs = pst.executeQuery();
     insumos = new ArrayList<>();
     result = new ArrayList<>();
     while (rs.next()) {

         oc= new OrdenCompra();
         oc.setIdOrdenCompra(rs.getInt(1));
          ProveedorDTO prov = new ProveedorDTO();
         prov.setNombre(rs.getString(2));
         oc.setProveedor(prov);
         InsumosDTO ins = new InsumosDTO();
         ins.setInsumo(rs.getString(3));
         insumos.add(ins);
         oc.setCantidad(rs.getDouble(4));
         oc.setFechaOC(rs.getDate(5));
         oc.setInsumos(insumos);
         result.add(oc);
     }
```

- **PEDIDO**

En el caso de pedido tendremos el método para registrar los datos generales del pedido a continuación. En este además se le asignará su correlativo después de cargar los datos.

```
public PedidoDTO  saveRequest(PedidoDTO pdto)
{
    try
    {
        pst=con.prepareStatement("select fnc_registerrequest(?,?,?,?,?);");
        pst.setInt(1, pdto.getIdEmpleado());
        pst.setInt(2, pdto.getIdCliente());
        pst.setBigDecimal(3, pdto.getTotal());
        pst.setBoolean(4, pdto.isEstado());
        pst.setTimestamp(5, pdto.getFechaEntrega());
        rs=pst.executeQuery();
        while(rs.next())
        {
            pdto=new PedidoDTO();
            pdto.setIdPedido(rs.getInt(1));
        }
```

Todas las consultas se harán cargando un arreglo PEDIDO, estas podrán ser por fecha, por los

atendidos, los no atendidos y los pedidos ingresados recientemente. A continuación los métodos utilizados.

Búsqueda por rango de fechas seleccionadas:

```
public ArrayList<PedidoDTO> getRequestorderBydateBetween(String str1, String str2)
{
    ArrayList<PedidoDTO> peddto= new ArrayList<PedidoDTO>();
    try
    {
        pst=con.prepareStatement("select * from fnc_getrequest_by_datebetween ('"
                +str1+"','"+str2+"');");
        rs=pst.executeQuery();
        while(rs.next())
        {
            peddto.add(new PedidoDTO(rs.getInt(1), rs.getInt(2),
                    rs.getInt(3),rs.getBigDecimal(4), rs.getString(5),
                    rs.getBoolean(6), rs.getTimestamp(7)));
        }

    }catch(SQLException w)
    {
    }
    finally
    {
        close();
    }
    return peddto;
}
```

Búsqueda de pedidos atendidos:

```
public ArrayList<PedidoDTO> getRequestorderByAtendidos()
{
    ArrayList<PedidoDTO> peddto= new ArrayList<PedidoDTO>();
    try
    {
        pst=con.prepareStatement("select * from fnc_selectpendientsrequestbyatendidos();");
        rs=pst.executeQuery();
        while(rs.next())
        {
            peddto.add(new PedidoDTO(rs.getInt(1), rs.getInt(2),
                    rs.getInt(3),rs.getBigDecimal(4), rs.getString(5),
                    rs.getBoolean(6), rs.getTimestamp(7)));
        }

    }catch(SQLException w)
    {
    }
    finally
    {
        close();
    }
    return peddto;
}
```

Búsqueda de pedidos No atendidos:

```
public ArrayList<PedidoDTO> getRequestorderByNoAtendidos()
{
    ArrayList<PedidoDTO> peddto= new ArrayList<PedidoDTO>();
    try
    {
        pst=con.prepareStatement("select * from fnc_selectpendientsrequestbynoatendidos();");
        rs=pst.executeQuery();
        while(rs.next())
        {
            peddto.add(new PedidoDTO(rs.getInt(1), rs.getInt(2),
                    rs.getInt(3),rs.getBigDecimal(4), rs.getString(5),
                    rs.getBoolean(6), rs.getTimestamp(7)));
        }

    }catch(SQLException w)
    {
    }
    finally
    {
        close();
    }
    return peddto;
}
```

Para atender los pedidos usaremos el siguiente método

```
public boolean AtenderRequest(PedidoDTO p){
    boolean ban = true;
    try
    {
        pst=con.prepareStatement("select fnc_atenderrequest("+
                p.getIdPedido()+", "+p.isEstado()+");");
        rs= pst.executeQuery();
        while(rs.next()){

        }JOptionPane.showMessageDialog(null,
                "PEDIDO ATENDIDO", "MENSAJE DE CONFIRMACIÓN",
                JOptionPane.INFORMATION_MESSAGE);
        ban= true;
    }catch(SQLException e){
            ban= false;
                JOptionPane.showMessageDialog(null,
                  "EL PEDIDO YA HA SIDO ATENDIDO", "ADVERTENCIA",
                  JOptionPane.WARNING_MESSAGE);
    }
    finally{

        close();
    }
    return ban;
```

- **DETALLEPEDIDODAO**

En esta clase se trabajarán los productos que formarán parte de los pedidos realizados por los clientes.

La siguiente clase retorna el pedido detallado, con su cliente que se asociará con los datos generales de la clase PEDIDO.

```
public Map<String, Object> getproductobyPedido(DetallePedidoDTO ped)
{
    Map<String, Object> mapa = new HashMap<>();
    ArrayList<ProductoDTO> arpdto= new ArrayList<ProductoDTO>();
    ped.setArProducto(arpdto);
    ClienteDTO cdto = new ClienteDTO();
    try
    {
        pst= con.prepareStatement(
            "select * from view_showproducts_on_request_client_person_details "
            + "where id_pedido=" +ped.getIdPedido()+";");
        rs=pst.executeQuery();
        while(rs.next())
        {
            cdto.setIdCliente(rs.getInt(2));
            cdto.setIdPersona(rs.getInt(3));
            cdto.setNombre(rs.getString(4));
            cdto.setApellidos(rs.getString(5));
            ped.getArProducto().add(new ProductoDTO(rs.getInt(6),rs.getString(7),
                                    rs.getInt(8), rs.getBigDecimal(9)));
        }
    }catch(SQLException e)
    {
        e.printStackTrace();
    }
    finally
    mapa.put("detalle", ped);
    mapa.put("cliente", cdto);
    return mapa;
}
```

- **PRODUCTODAO**

Para la inserción de los productos usaremos el siguiente método:

```
public void insertProduct(ProductoDTO pdto)
{
   try
   {
       System.out.println(pdto.getProducto()+" "
       +pdto.getCantidad()+","+pdto.getPrecio()+","+pdto.getIdCtegoria());
        pst=con.prepareStatement("select fnc_insertproduct('"
            +pdto.getProducto()+"','"+pdto.getCantidad()+","
            +pdto.getPrecio()+","+pdto.getIdCtegoria()+
            ");");
        rs= pst.executeQuery();
        while(rs.next())
        {
            JOptionPane.showMessageDialog(null, rs.getString(1));
        }
```

Para la búsqueda y/o muestra de productos primero se necesita cargarlos en un arreglo así como

se muestra a continuación.

```
public ArrayList<ProductoDTO> obtenerProductos()
{
    ArrayList<ProductoDTO> apdto= new ArrayList();
    try
    {
        pst=con.prepareStatement("select * from producto");
        rs=pst.executeQuery();
        while(rs.next())
        {
            apdto.add(new ProductoDTO(rs.getInt(1),
                    rs.getString(2), rs.getInt(3),
                    BigDecimal.valueOf(rs.getDouble(4)), rs.getInt(5)));
        }
    }catch(SQLException e)
    {}
    finally
    {   close(); }
    return apdto;
```

El detalle del método de búsqueda de producto, el cual devolverá un objeto producto, que será cargado en la ventana de la aplicación.

```
public ProductoDTO searchProduct(Integer c)
{
    String temp="";
    Object arrayTemp[]=null;
    ProductoDTO pdto= null;
    try
    {
        pst= con.prepareStatement("select fnc_searchproduct("+c+");");
        rs=pst.executeQuery();
        while(rs.next())
        {
            temp=rs.getString(1);
            temp= temp.trim();
            temp= temp.replace("\"", "");
            temp= temp.replace("(", "");
            temp= temp.replace(")", "");
            arrayTemp=temp.split(",");
            JOptionPane.showMessageDialog(null,temp);
            pdto=new ProductoDTO(Integer.parseInt((String) arrayTemp[0]),
                    String.valueOf(arrayTemp[1]), Integer.parseInt((String)arrayTemp[2]),
                    BigDecimal.valueOf(Double.parseDouble((String) arrayTemp[3])),
                    Integer.parseInt((String)arrayTemp[4]));
```

```
        }
    }catch(SQLException g)
    {

     }catch(NullPointerException g)
     {
         JOptionPane.showMessageDialog(null,"Producto no encontrado");
     }
     catch(ArrayIndexOutOfBoundsException e)
     {
         JOptionPane.showMessageDialog(null, "EL PRODUCTO NO EXISTE");
     }
     finally
     {
         close();
     }
     return pdto;
}
```

La búsqueda de productos por categorías se llamará en DAO:

```
public List<ProductoDTO> obtenerProductosByCategoria(CategoriaDTO catdto) {
    ArrayList<ProductoDTO> apdto= new ArrayList();
    try
    {
        pst=con.prepareStatement("select * from fnc_getallproductsbycategoria("
                +catdto.getIdCategoria()+");");
        rs=pst.executeQuery();
        while(rs.next())
        {
            apdto.add(new ProductoDTO(rs.getInt(1),
                    rs.getString(2), rs.getInt(3),
                    BigDecimal.valueOf(rs.getDouble(4)), rs.getInt(5)));
        }
    }catch(SQLException e)
    { }
    finally
    {  close();}
    return apdto;
```

Actualizar el producto cargará los nuevos datos al registro seleccionado.

```
public void updateProduct(ProductoDTO pdto) {
    try
    {
        pst=con.prepareStatement("select * from fnc_updateproduct("
                +pdto.getIdProducto()+",'"+pdto.getProducto()+"',"
                +pdto.getCantidad()+","+pdto.getPrecio()+","+pdto.getIdCtegoria()+
                ");");
        rs= pst.executeQuery();
        while(rs.next())
        {
            JOptionPane.showMessageDialog(null, rs.getString(1));
        }
    }
    catch(SQLException e)
    {
        JOptionPane.showMessageDialog(null, e.getMessage());
```

- **USUARIODAO**

Se creará el usuario con un procedimiento simple de registro.

```
public void createUser(UsuarioDTO usdto, Integer id){

    try{
        st=con.createStatement();
        rs= st.executeQuery("select * from fnc_registeruser("
                +id+", '"+usdto.getContrasenha()+"','"+
                usdto.getUsuario()+"',"+usdto.isEstado()+",'"
                +usdto.getNombre()+"',"+usdto.isIdRol()+");");
        rs.next();
        JOptionPane.showMessageDialog(null,
        "USUARIO CREADO CORRECTAMENTE", "CONFIRMACION",
        JOptionPane.INFORMATION_MESSAGE);
    }catch(SQLException e){
     JOptionPane.showMessageDialog(null, "ERROR DE OPERACION",
            "ADVERTENCIA", JOptionPane.WARNING_MESSAGE);

    }
    finally{
    close();
    }
```

Al ingresar al sistema se tendrá que validar por seguridad que el usuario ingresado sea el correcto, para esto se ha trabajado el método en DAO llamando a la función fnc_getuser que permitirá validar los datos, en ella se implementará el método de descencriptación de la contraseña para mayor seguridad.

```
public UsuarioDTO getUsuario(UsuarioDTO usdto)
{
    try
    {
        System.out.println("usuario ing:"+usdto.getUsuario());
        st=con.createStatement();
        rs=st.executeQuery("select * from fnc_getuser('"
              +usdto.getUsuario()+"','"+usdto.getContrasenha()
              +"','"+usdto.isIdRol()+"');");
        if(rs.next())
        {
            udto= new UsuarioDTO();
            udto.setId(rs.getInt(1));
            udto.setContrasenha(rs.getString(2));
            udto.setUsuario(rs.getString(3));
            udto.setNombre(rs.getString(5));
            udto.setEstado(rs.getBoolean(6));
        }

    }catch(SQLException e)
    { }
    return udto;
}
```

- **PERSONADAO**

Para el registro de una persona, cargaremos los datos ingresados y en el mismo método se le asignará el correlativo respectivo.

```
public Integer registerPerson(EmpleadoDTO p){
    Integer id_per=0;
    try{
    pst = con.prepareStatement("select * from fnc_create_person('"
            +p.getNombre()+"','"+p.getApellidos()+ "','"
            +p.getDireccion()+"','"+p.getDni()+"','"+p.getRuc()+"');");
     rs = pst.executeQuery();
    while(rs.next()){
        id_per=rs.getInt(1);
    }
    }catch(SQLException e){
    e.printStackTrace();
    }
    finally{
    close();
    }
    return id_per;
}
```

Para la búsqueda se realizará una consulta simple desde el método de java.

```
public PersonaDTO obtenerPersona(UsuarioDTO u)
{
    PersonaDTO pdto=null;
    //JOptionPane.showMessageDialog(null, u.getUsuario());
    try
    {
        pst=con.prepareStatement(
                "select * from persona where id_persona=(select id_persona "
                    + "from empleado where id_usuario=(select id from usuario where usuario ='"
                    +u.getUsuario()+"' and contrasenha='"+u.getContrasenha()+"'));");
        rs=pst.executeQuery();
        while(rs.next())
        {
            pdto= new PersonaDTO(rs.getInt(1), rs.getString(2), rs.getString(3),
                    rs.getString(4), rs.getString(5), rs.getString(6));
        }
    }catch(SQLException e)
    {

    }
    return pdto;
```

- **CLIENTEDAO**

Para guardar un cliente se trabaja capturando datos y agregándolo al objeto.

```
public Integer save_cliente(ClienteDTO cliente_param, String telefonos) {
    Integer result = null;
    try {

        pst = con.prepareStatement("select * from fnc_register_cliente(?,?,?,?,?,?);");
        pst.setString(1, cliente_param.getNombre());
        pst.setString(2, cliente_param.getApellidos());
        pst.setString(3, cliente_param.getDireccion());
        pst.setString(4, cliente_param.getDni());
        pst.setString(5, cliente_param.getRuc());
        pst.setString(6, telefonos);

        rs = pst.executeQuery();
        while (rs.next()) {
            result = rs.getInt(1);
        }
        return result;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } finally {
        close();
    }
```

- **PROVEEDORES**

La búsqueda de proveedores también se hará según su categoría. Esta se está encapsulando en el método a continuación:

```
public List<ProveedorDTO>  getProveedoresByCategoria(int codigoCategoria) {

List<ProveedorDTO> proveedores ;

try {
    proveedores = new ArrayList<>();

    pst = con.prepareStatement("select * from fnc_proveedor_by_categoria(?);");
    pst.setInt(1, codigoCategoria);
    rs = pst.executeQuery();
    while (rs.next()) {
        ProveedorDTO proveedor = new ProveedorDTO();
        proveedor.setIdProveedor(rs.getInt(1));
        proveedor.setNombre(rs.getString(2));
        proveedor.setApellidos(rs.getString(3));
        proveedor.setDireccion(rs.getString(4));
        //cliente.setDni(rs.getString(5));
        proveedor.setRuc(rs.getString(5));
        proveedor.setTipoPago(rs.getBoolean(6));

        proveedores.add(proveedor);
    }

    return proveedores;
```

Cabe resaltar, como ya se ha detallado anteriormente que en el presente informe sólo se ha considerado los métodos más relevantes o que compilan operacines que en otras clases se repiten o son parecidos. El código restante será presentado en anexos.

Las funciones mencionadas en el código anteriormente mostrado se presentarán en el siguiente punto.

5.1.3 Reportes

Los reportes estarán trabajados con JasperReport, el método para que se desarrollen las llamadas a los reportes estará detallado en el sistema en la clase ReportGenerator. Y sus métodos estarán detallados en los métodos de esta, a continuación un ejemplo:

```
public static void generarReporteInsumos() {
    conectar();

    try {
        String path = "C:\\Users\\USUARIO\\Documents\\NetBeansProjects"
                + "\\SysPanaderia\\src\\reportes\\reportInsumos.jasper";
        JasperReport jr = (JasperReport) JRLoader.loadObjectFromFile(path);
        JasperPrint jp = JasperFillManager.fillReport(jr, null, con);
        JasperViewer jv = new JasperViewer(jp, false);
        jv.setVisible(true);
        jv.setTitle("REPORTE DE INSUMOS DE ALMACEN");
    } catch (JRException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    } finally {
        try {
            con.close();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }
}
```

5.1.4 Temporizadores y Quarz

Para las alertas se ha trabajado con Quarz, que permitirá manejar el tiempo en que esta se mostrará y así recordar al usuario la función principal que es Generar la Orden de Compra cuando los insumos estén por debajo de su cantidad mínima.

En primer lugar tendremos un temporizador, esta clase se ejecutará con un hilo, al mismo tiempo que el sistema, y enviará la alerta según el reporte de insumos

```
public class Temporizador extends TimerTask {
    ArrayList<InsumosDTO> arIndto;
InsumosDAO idao= new InsumosDAO();
```

El método que ejecuta la función antes mencionada verificará el día, y enviará la alerta de lunes a sábado, como se muestra en la condición múltiple.

Si el Stock se encuentra por debajo de su condición mínima mostrará un mensaje de advertencia.

```
public void run() {
    Calendar diaActual = Calendar.getInstance();

    switch(diaActual.get(Calendar.DAY_OF_WEEK)){
        case 1: break;
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
            arIndto= idao.getAllInputsMinimum();
            if(arIndto.size()>0){

                Integer valor =JOptionPane.showConfirmDialog(null,
                "Insumos por debajo del stock mínimo", "ADVERTENCIA",
                JOptionPane.WARNING_MESSAGE);
                Interceptor.llamar(valor);
```

Si el mensaje es aceptado llama al método LLAMAR de la clase interceptor la cual se encarga de llamar a la ventana Generar Orden de Compra y así poder generarla y cumplir con el requerimiento.

```
public static  void llamar(Integer valor){

    if(valor==JOptionPane.YES_OPTION){
    ViewOrdenCompra voc= new ViewOrdenCompra();
    jdpContent.add(voc);
    voc.setSize(jdpContent.getSize());
    voc.setVisible(true);
    }
```

Seguidamente se tendrá una clase Quartz que permitirá enviar la alerta cada cierto tiempo determinado.

```
public class Quartz {

    Date horaMensaje = new Date(System.currentTimeMillis());
    Calendar c;
```

El constructor de esta clase permitirá controlar el tiempo en que aparece el mensaje de advertencia, desde aquí se controlará la ventana.

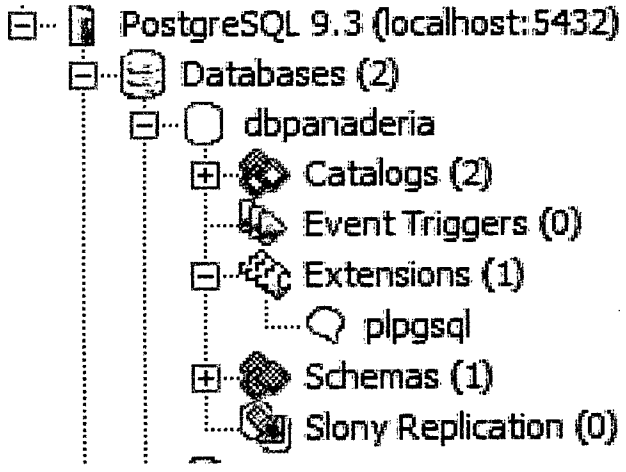
```
public Quartz(){

    c = Calendar.getInstance();
    c.setTime(horaMensaje);
    System.out.println(c.get(Calendar.DAY_OF_WEEK));
    System.out.println(c.get(Calendar.HOUR_OF_DAY));
    System.out.println("minuto"+c.get(Calendar.MINUTE));
    if (c.get(Calendar.HOUR_OF_DAY) >= 18 && c.get(Calendar.MINUTE)>=59) {
                c.set(Calendar.DAY_OF_YEAR, c.get(Calendar.DAY_OF_YEAR) + 1);
            }

    c.set(Calendar.HOUR_OF_DAY, 0);
            c.set(Calendar.MINUTE, 31);
            c.set(Calendar.SECOND, 0);

            horaMensaje = c.getTime();
            System.out.println(horaMensaje);
            System.out.println(c.get(Calendar.DAY_OF_WEEK));
            int tiempoRepeticion = 30000;

            Timer temporizador = new Timer();
            temporizador.schedule(new Temporizador(), horaMensaje, tiempoRepeticion);
```

5.2 DESARROLLO DE LA BASE DE DATOS

En este punto se describe los puntos trabajados en la base de datos. Partiendo desde las tablas, vistas, secuencias, funciones, etc.

El sistema de gestión de base de datos que se eligió es POSTGRSQL, que es un orientado a objetos libre de código abierto.

Se ha incluido el PL/PgSQL como lenguaje complementario ya que permite realizar cálculos complejos y crear nuevos tipos de datos de usuario y así ejecutar una base de datos más organizada y con más funciones que permitan obtener mejores resultados.

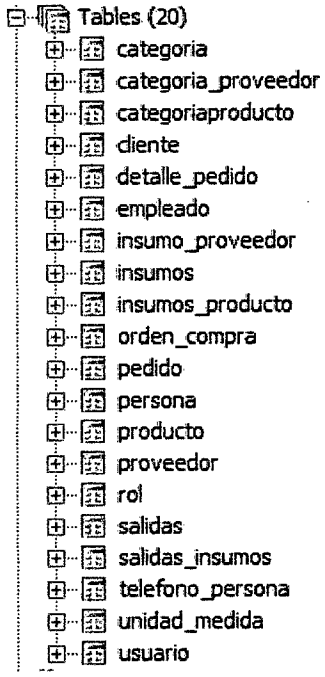


5.2.1. TABLAS:

La base de datos del sistema consta de 20 tablas normalizadas.. El procedimiento para crear cada una de las tablas será el siguiente, y la clave primario será registrada en id_insumo, y así en cada una de las tablas.

```
CREATE TABLE insumos
(
  id_insumo serial NOT NULL,
  insumo character varying(30),
  um character(2),
  minima numeric(6,3),
  cantidad numeric(6,3),
  CONSTRAINT insumo_pkey PRIMARY KEY (id_insumo)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE insumos
  OWNER TO postgres;
```

Lista de tablas a continuación, con las cuales se trabajará la base de datos.



5.2.2. FUNCIONES:

- **fcn_createprimaryproduct**

```
CREATE OR REPLACE FUNCTION fcn_createprimaryproduct(character varying, character, numeric, numeric)
  RETURNS integer AS
$BODY$
declare
ins alias for $1;
unm alias for $2;
min alias for $3;
cant alias for $4;
id_ins integer;
begin
                insert into insumos (insumo, um, minima, cantidad) values (ins, unm, min, cant);
                select into id_ins last_value from insumos_id_insumo_seq;
                return id_ins;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fcn_createprimaryproduct(character varying, character, numeric, numeric)
  OWNER TO postgres;
```

- **fnc_atenderrequest**

```
CREATE OR REPLACE FUNCTION fnc_atenderrequest(integer, boolean)
  RETURNS void AS
$BODY$
declare
id alias for $1;
ate alias for $2;
ban boolean;
begin
        select into ban estado from pedido where id_pedido = id;
        if ban = true then
                RAISE EXCEPTION 'EL PEDIDO YA HA SIDO ATENDIDO';
        else
                update pedido set estado = ate where id_pedido = id;

        end if;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_atenderrequest(integer, boolean)
  OWNER TO postgres;
```

- **fnc_create_person**

```
CREATE OR REPLACE FUNCTION fnc_create_person(character varying,
character varying, character varying, character, character)
  RETURNS integer AS
$BODY$
declare
nom alias for $1;
ap alias for $2;
dir alias for $3;
dni alias for $4;
rc alias for $5;
id integer;
begin
        insert into persona(nombre, apellidos, direccion, dni, ruc)
        values (nom, ap, dir, dni, rc);

        select into id max(id_persona) from persona;
        return id;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_create_person(character varying, character varying,
character varying, character, character)
  OWNER TO postgres;
```

- **fnc_filteroutputs()**

```
CREATE OR REPLACE FUNCTION fnc_filteroutputs()
  RETURNS SETOF view_employesoutputsperson AS
$BODY$
declare
tabla view_employesoutputsperson%rowtype;
begin
        for tabla in select * from view_employesoutputsperson order by
        fecha asc loop return next tabla;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_filteroutputs()
  OWNER TO postgres;
```

- **fnc_getallprovider**

```
CREATE OR REPLACE FUNCTION fnc_getallprovider()
  RETURNS SETOF view_proveedor_persona AS
$BODY$
declare
tabla view_proveedor_persona%rowtype;
begin
        for tabla in select * from view_proveedor_persona loop return next tabla;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_getallprovider()
  OWNER TO postgres;
```

- **fnc_getnextidcliente**

```
CREATE OR REPLACE FUNCTION fnc_getnextidcliente()
  RETURNS integer AS
$BODY$
declare
        id integer;
begin
        select into id last_value FROM cliente_id_cliente_seq;

        return id;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_getnextidcliente()
  OWNER TO postgres;
```

- **fnc_getrequest_by_datebetween**

```
CREATE OR REPLACE FUNCTION fnc_getrequest_by_datebetween(timestamp
without time zone, timestamp without time zone)

  RETURNS SETOF pedido AS
$BODY$
declare
t1 alias for $1;
t2 alias for $2;
tabla pedido%rowtype;
begin
        for tabla in select * from pedido where fecha_entrega
        between t1 and t2 loop return next tabla;
        end loop;

end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_getrequest_by_datebetween(timestamp without time zone,
 timestamp without time zone)

  OWNER TO postgres;
```

- **fnc_insertproduct**

```
CREATE OR REPLACE FUNCTION fnc_insertproduct(character varying, integer,
numeric, integer)
  RETURNS character varying AS
$BODY$
declare
prod alias for $1;
cant alias for $2;
prec alias for $3;
idcat alias for $4;
msj varchar(20);
tabla producto%rowtype;
begin
        select into tabla * from producto where producto = lower(prod) a
        nd id_categoria=idcat;
        if found then
                raise exception 'El producto ya existe';
        else
                insert into producto ( producto, cantidad, precio, id_categoria)
                values (prod, cant, prec, idcat);

                msj= 'Inserción exitosa';
        end if;
        return msj;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_insertproduct(character varying, integer, numeric, integer)
  OWNER TO postgres;
```

- **fnc_list_all_cliente**

```
CREATE OR REPLACE FUNCTION fnc_list_all_cliente()
  RETURNS TABLE(codcliente_bd integer, nombre_bd character varying, apellidos_bd
  character varying, direccion_bd character varying, dni_bd character, ruc_bd character) AS
$BODY$
BEGIN
                RETURN QUERY
                SELECT c.id_cliente,p.nombre, p.apellidos, p.direccion, p.dni, p.ruc
                FROM cliente c
                INNER JOIN persona p
                ON c.id_persona=p.id_persona;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_list_all_cliente()
  OWNER TO postgres;
```

- **fnc_proveedor_by_categoria**

```
REATE OR REPLACE FUNCTION fnc_proveedor_by_categoria(IN inidcategoria integer)
  RETURNS TABLE(codproveedor_bd integer, nombre_bd character varying,
  apellidos_bd character varying, direccion_bd character varying, ruc_bd character,
  tipopago_bd boolean, telefonos character varying) AS
$BODY$
DECLARE
        varraytelefonos character varying;
        r record;
        table_record record;
        idprov integer;

BEGIN
        varraytelefonos:='';

        for r in select id_proveedor from categoria_proveedor where id_categoria=inidcategoria loop

        FOR table_record IN SELECT tp.telefono FROM telefono_persona tp inner join proveedor pr ON
                tp.id_persona = pr.id_persona  WHERE pr.id_proveedor = r.id_proveedor LOOP

                --EXECUTE 'SELECT count(*) FROM ' || table_record.tablename INTO nbRow;
                idprov:=r.id_proveedor;
                select concat_ws(',', varraytelefonos, table_record.telefono) INTO varraytelefonos;

                -- Do something with nbRow
        END LOOP;
```

```
            RETURN QUERY
            SELECT pr.id_proveedor,p.nombre, p.apellidos, p.direccion, p.ruc,pr.tipo_pago,
            varraytelefonos
            FROM proveedor pr
            INNER JOIN persona p
            ON pr.id_persona=p.id_persona
            where pr.id_proveedor = idprov;
        END LOOP;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_proveedor_by_categoria(integer)
  OWNER TO postgres;
```

- **fnc_register_cliente**

```
CREATE OR REPLACE FUNCTION fnc_register_cliente(innombre character varying,
inapellidos character varying, indireccion character varying, indni character varying,
inruc character varying, intelefonos character varying)
  RETURNS integer AS
$BODY$
        DECLARE
        vcodigopersona integer;
        vresult integer;
        vindex integer;
        vtelefonos character varying[];

BEGIN
        vresult:=0;
        vindex:=1;
        vtelefonos:=string_to_array(intelefonos,',');
        RAISE NOTICE 'FUERA DEL IF';


        IF ((SELECT COUNT(*) FROM persona where dni = indni AND indni<>'')=0 and (SELECT COUNT(*)
        FROM persona where ruc = inruc AND inruc <>'')=0 )THEN
        --INSERTAR PERSONA NO DEBE EXISIIIR YA UNA PERSONA CON EL MISMO DNI O RUC
        RAISE NOTICE 'dENTRO DEL PRIMER IF';
                INSERT INTO persona(
                                nombre, apellidos, direccion, dni, ruc)
                               VALUES (innombre, inapellidos, indireccion , indni, inruc);

                SELECT currval('persona_id_persona_seq') into vcodigopersona;
```

```
                IF ((select COUNT(*) FROM cliente where id_persona=vcodigopersona)=0) THEN
                        INSERT INTO cliente(
                         id_persona)
                        VALUES (vcodigopersona);
                    --INSERTAR TELEFONOS
                    WHILE (vindex <= array_length(vtelefonos, 1)) LOOP
                            INSERT INTO telefono_persona(id_persona, telefono)
                            VALUES(vcodigopersona,vtelefonos[vindex]);
                            vindex:=vindex+1;
                    END LOOP;
                    vresult:=1;
                    return vresult;
                END IF;
        ELSE
                vresult:=0;
                return vresult;
        END IF;

        return vresult;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_register_cliente(character varying, character varying, character varying,
character varying, character varying, character varying)
  OWNER TO postgres;
```

- **fnc_searchemployes**

```
CREATE OR REPLACE FUNCTION fnc_searchemployes()
  RETURNS SETOF view_employes_person_details AS
$BODY$
declare
tabla view_employes_person_details%rowtype;
begin
        for tabla in select * from view_employes_person_details loop return next tabla;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_searchemployes()
  OWNER TO postgres;
```

- **fnc_selectallprimaryproducts**

```
CREATE OR REPLACE FUNCTION fnc_selectallprimaryproducts()
  RETURNS SETOF view_primaryproduct_primaryproductprovider_provider_persona AS
$BODY$
declare
tabla view_primaryproduct_primaryproductprovider_provider_persona%rowtype;
begin
        for tabla in select * from
        view_primaryproduct_primaryproductprovider_provider_persona
        loop return next tabla;

        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_selectallprimaryproducts()
  OWNER TO postgres
```

- **fnc_selectpendientsrequestbyatendidos**

```
CREATE OR REPLACE FUNCTION fnc_selectpendientsrequestbyatendidos()
  RETURNS SETOF pedido AS
$BODY$
declare
tabla pedido%rowtype;
begin
        for tabla in select * from pedido where estado = true order by
        fecha_pedido asc loop return next tabla;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_selectpendientsrequestbyatendidos()
  OWNER TO postgres;
```

- **fnc_selectpendientsrequestbydate**

```
CREATE OR REPLACE FUNCTION fnc_selectpendientsrequestbydate()
  RETURNS SETOF pedido AS
$BODY$
declare
tabla pedido%rowtype;
begin
        for tabla in select * from pedido order by fecha_pedido
        desc loop return next tabla;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_selectpendientsrequestbydate()
  OWNER TO postgres;
```

- **fnc_update_cliente**

```
CREATE OR REPLACE FUNCTION fnc_update_cliente(incodigocliente integer,
innombre character varying, inapellidos character varying, indireccion character varying,
indni character varying, inruc character varying, intelefonos character varying)
  RETURNS integer AS
$BODY$
        DECLARE
        vcodigopersona integer;
        vresult integer;
        vindex integer;
        vtelefonos character varying[];

BEGIN
        vresult:=0;
        vindex:=1;
        vtelefonos:=string_to_array(intelefonos,',');
        RAISE NOTICE 'FUERA DEL IF';
        SELECT p.id_persona into vcodigopersona from persona p inner join cliente c on
        c.id_persona = p.id_persona where c.id_cliente = incodigocliente;

        IF ((SELECT COUNT(*) FROM persona where id_persona=vcodigopersona)=1 )THEN
        --MODIFICAR PERSONA NO DEBE EXISITIR YA UNA PERSONA CON EL MISMO DNI O RUC
        RAISE NOTICE 'dENTRO DEL PRIMER IF';

                UPDATE persona SET nombre= innombre, apellidos= inapellidos, direccion=indireccion,
                 dni= indni, ruc=inruc where id_persona = vcodigopersona ;


                IF ((select COUNT(*) FROM cliente where id_persona=vcodigopersona)=1) THEN

                        WHILE (vindex <= array_length(vtelefonos, 1)) LOOP
                                if ((SELECT COUNT(*) FROM telefono_persona where
                                        telefono=vtelefonos[vindex])=0)then
                                        INSERT INTO telefono_persona(id_persona, telefono)
                                        VALUES(vcodigopersona,vtelefonos[vindex]);
                                else
                                        UPDATE telefono_persona SET  telefono = vtelefonos[vindex]
                                        WHERE id_persona = vcodigopersona;
                                end if;

                                vindex:=vindex+1;
                        END LOOP;
                        vresult:=1;
                        return vresult;
                END IF;
        ELSE
                vresult:=0;
                return vresult;
        END IF;

        return vresult;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_update_cliente(integer, character varying, character varying, character varying,
character varying, character varying, character varying)
  OWNER TO postgres;
```

- **fnc_update_request_if_date_expired**

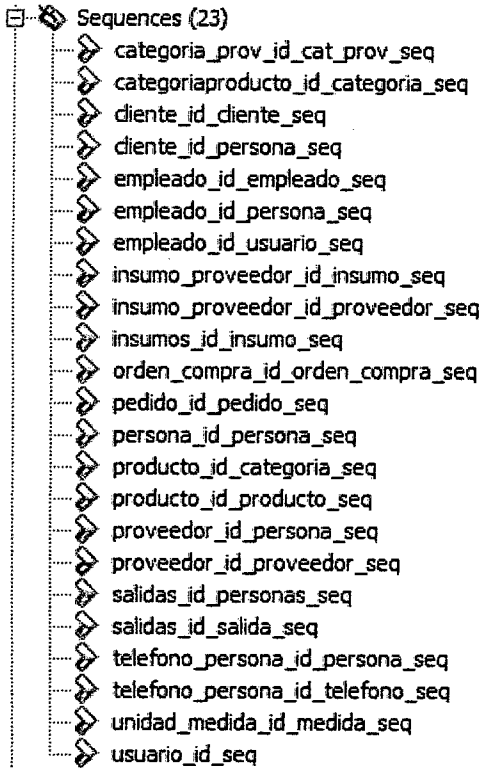
```
CREATE OR REPLACE FUNCTION fnc_update_request_if_date_expired()
  RETURNS SETOF pedido AS
$BODY$
declare
tabla pedido%rowtype;
begin
        for tabla in select * from pedido loop  return next tabla;
        if tabla.fecha_entrega < current_date then
                update pedido set anulado = true where pedido.id_pedido=tabla.id_pedido;
        else
                update pedido set anulado = false where pedido.id_pedido=tabla.id_pedido;
        end if;
        end loop;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION fnc_update_request_if_date_expired()
  OWNER TO postgres;
```

- **fnc_registeruser**

```
CREATE OR REPLACE FUNCTION fnc_registeruser(integer, character varying,
character varying, boolean, character varying, boolean)
  RETURNS void AS
$BODY$
declare
id_em alias for $1;
contr alias for $2;
us alias for $3;
est alias for $4;
nomb alias for $5;
id_r alias for $6;
id_us integer;
begin
        insert into usuario (contrasenha, usuario, estado, nombres, id_rol)
        values (md5(contr||us), us, est, nomb, id_r);
        select into id_us max(id) from usuario;
        update empleado set id_usuario= id_us where id_empleado=id_em;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION fnc_registeruser(integer, character varying, character varying,
boolean, character varying, boolean)
  OWNER TO postgres;
```

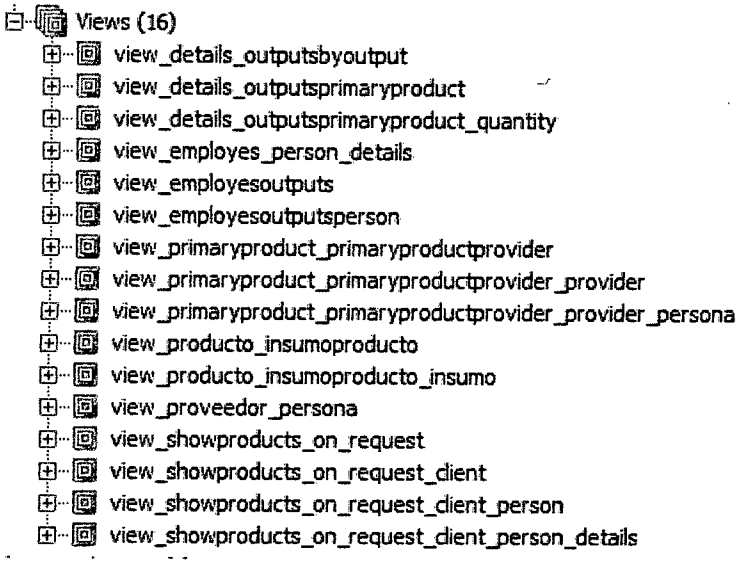
5.2.3. SECUENCIAS

```
CREATE SEQUENCE categoria_prov_id_cat_prov_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 3
  CACHE 1;
ALTER TABLE categoria_prov_id_cat_prov_seq
  OWNER TO postgres;
```



5.2.4. VISTAS

```
CREATE OR REPLACE VIEW view_primaryproduct_primaryproductprovider AS
 SELECT i.id_insumo AS codigo,
    i.insumo,
    i.um,
    i.minima,
    i.cantidad,
    ip.id_proveedor,
    ip.precio
   FROM insumos i
     LEFT JOIN insumo_proveedor ip ON i.id_insumo = ip.id_insumo;

ALTER TABLE view_primaryproduct_primaryproductprovider
  OWNER TO postgres;
```



5.3 SEGURIDAD DE LA INFORMACIÓN

5.3.1.- Usuarios y Accesos:

La seguridad de la información es el conjunto de medidas preventivas y reactivas de las organizaciones y de los sistemas tecnológicos que permitan resguardar y proteger la información buscando mantener la confidencialidad, la disponibilidad e integridad de la misma

Para propósitos de seguridad, se crearon dos tipos de niveles de acceso. Estos niveles se subdividen en Administrador y Vendedor. Aquel usuario con acceso de Administrador tendrá acceso a todas las pantallas y procesos creados en el sistema. Por otro lado, aquel usuario que tenga un nivel de acceso de Vendedor solo podrá realizar consultas e ingresos de pedidos de clientes y datos de estos.

Esto nos ayudara a confiar en que no se harán cambios indebidos en el sistema sin el consentimiento del Administrador. Se evitara así posibles pérdidas de información y se proveerá un ambiente de seguridad para los usuarios en general, evitando pedidos no autorizados y un mal manejo del Inventario general de la Panadería.

Para asegurar la integridad de la información que es enviada desde el cliente y recibida por el servidor se hará uso del método hash MD5, es un algoritmo de reducción criptográfico, que permitirá codificar las contraseñas ingresadas al momento de registrarlas en la base de datos para así no poder manipular la información así se tenga acceso directo al gestor de base de datos.

5.3.2.- Backup y resguardo de información:

Una de las cosas más importantes de hacer es respaldar los datos en caso de que ocurra cualquier situación que dañe nuestro registro de datos.

Herramienta: pg_dump

Pg_dump es una herramienta de línea de comandos que permite hacer un respaldo de base de datos en el gestor utilizado en el presente sistema: Postgresql.

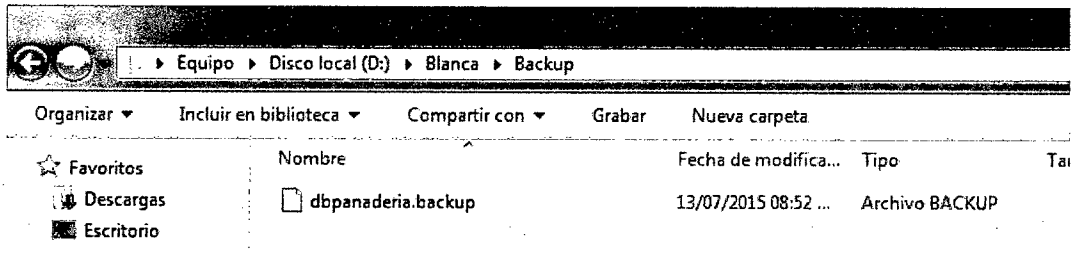
Permite hacer el volcado de datos en diferentes formatos escribiendo en un archivo las instrucciones SQL necesarias para hacer un respaldo de la base de datos.

Para ejecutar la función a diario sin que el usuario tenga acceso directo a código del sistema y/o de la base de datos se hará uso de **Quartz,** lo que permite programar una tarea, tal como se ha mostrado anteriormente en el presente informe.

Se tiene una base de datos llamada **dbpanaderia** y para un mejor manejo de los datos se respaldará en un archivo llamado **dbpanaderia.sql**, utilizando el comando:

pq_dump dbpanaderia > dbpanaderia.sql

Al ejecutar la función antes mencionada se crea un archivo comprimido que contiene toda la información de la base de datos guardándose en la dirección señalada:

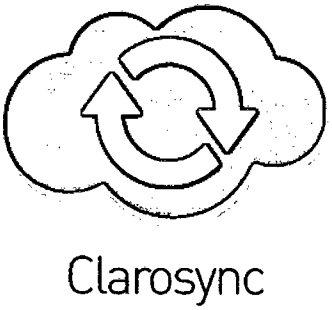


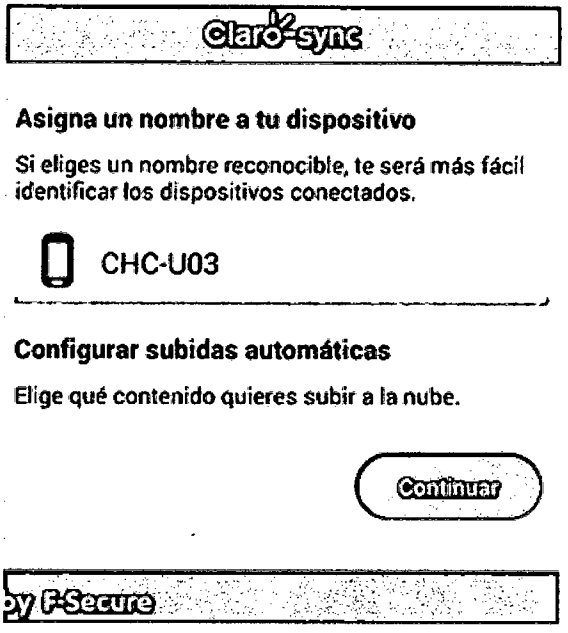
El sistema de gestión de logística para la panadería "Dos estrellas" es un sistema de información de escritorio, el cual tienen acceso a internet por pedido y/o algunas limitantes del usuario principal (administrador).

A esta limitante de internet en el dispositivo en que se ejecutará el software creado se ha propuesto utilizar, aparte de la PC de escritorio, un dispositivo móvil (Smartphone) de Gama baja media o alta.

La tecnología Bluetooth permite la transmisión de datos de manera inalámbrica entre dispositivos que cuenten Bluetooth, en el presente caso la PC y el dispositivo móvil esto permitirá al usuario ir a la carpeta donde está guardado el archivo comprimido y pasarlo mediante esta herramienta al dispositivo móvil (una vez sincronizado).

Una vez en el dispositivo móvil, haciendo uso de la tecnología actual, se subirá a la nube, que es un servicio gratuito que se ofrece en los distintos operadores.





5.4.- PRUEBAS DEL SISTEMA

Pruebas del funcionamiento del sistema, se medirá el tiempo de ejecución de procesos por cada uno de los módulos.

5.4.1. Acceso al sistema

- **Prueba 01.-** Prueba de ingreso al sistema y conexión con BD

N°	Descripción	Resultado
1	El usuario administrador ingresa al sistema	OK
2	El usuario vendedor ingresa al sistema	OK

Tabla 5.4.1.1 Acceso al sistema

5.4.2.- Módulo Archivo

- **Prueba 02.-** Prueba de mantenimiento de archivos con el proceso manual

N°	Descripción	Tiempo
1	Creación de clientes	1 min
2	Modificación de clientes	1 min
3	Búsqueda de clientes	5 min
4	Ingreso de productos	1 min
5	Modificación de productos	1 min
6	Búsqueda de productos	7 min
7	Ingreso de Proveedores	2 min
8	Búsqueda de proveedores	5 min
9	Modificación de proveedores	1 min
10	Listado de productos	15 min
	Tiempo total	**39 min**

Tabla 5.4.2.1. Prueba Archivo manual

Prueba 03.- Prueba de mantenimiento de archivos haciendo uso del sistema

N°	Descripción	Tiempo
1	Creación de clientes	1 min
2	Modificación de clientes	20 seg
3	Búsqueda de clientes	3 seg
4	Ingreso de productos	1 min
5	Modificación de productos	10 seg
6	Búsqueda de productos	2 seg
7	Ingreso de Proveedores	1.5 min
8	Búsqueda de proveedores	2 seg

9	Modificación de proveedores	20 seg
10	Listado de productos	2 seg
	Tiempo total	**4.48 min**

Tabla 5.4.2.2. Prueba Archivo sistema

Se han comparado los tiempos de desarrollo de las dos pruebas anteriores:

Prueba	Descripción	Tiempo (Minutos)
Prueba 02	Prueba de funcionamiento del procedimiento manual	39
Prueba 03	Prueba de funcionamiento del procedimiento con el sistema	4.48

Tabla 5.4.2.3. Comparación de datos Archivo

5.4.3 Módulo Pedidos:

- **Prueba 04.** Prueba de funcionamiento del sistema ejecutando los procesos de pedidos manualmente.

N°	Descripción	Tiempo
1	Creación de pedidos	2 min
2	Búsqueda de pedidos por fecha de entrega	4 min
3	Búsqueda de pedidos por estado	4 min
	Tiempo total	12 min

Tabla 5.4.3.1 Prueba Pedidos manual

- **Prueba 05.** Prueba de funcionamiento de los procesos de pedidos haciendo uso del sistema:

N°	Descripción	Tiempo
1	Creación de pedidos	1.5 min
2	Búsqueda de pedidos por fecha de entrega	2 seg
3	Búsqueda de pedidos por estado	2 seg
	Tiempo total	1.57 min

Tabla 5.4.3.2 Prueba Pedidos sistema

Comparación de tiempos delas pruebas del sistema:

Prueba	Descripción	Tiempo (Minutos)
Prueba 04	Prueba de funcionamiento del procedimiento manual	12
Prueba 05	Prueba de funcionamiento del procedimiento con el sistema	1.57

Tabla 5.4.3.3 Comparación de resultados pedidos

5.4.4 Modulo de Almacén

- **Prueba 06.** Pruebas de mantenimiento de Almacén con el procedimiento manual:

N°	Descripción	Tiempo
1	Creación de insumo	1 min
2	Modificación de insumo	1 min
3	Búsqueda de insumo	5 min
4	Listado de insumos	20 min
5	Creación de salida	5 min
6	Visualización de salida	2 seg
	Tiempo total	27.12 min

Tabla 5.4.4.1 Pruebas almacén manual

- **Prueba 07.-** Pruebas de mantenimiento de almacén con el sistema

N°	Descripción	Tiempo
1	Creación de insumo	1 min
2	Modificación de insumo	10 seg
3	Búsqueda de insumo	2 seg
4	Listado de insumos	2 seg
5	Creación de salida	1 min
6	Visualización de salida	2 seg
	Tiempo total	2.27 min

Tabla 5.4.4.2 Pruebas almacén sistema

Los resultados y la comparación de tiempos:

Prueba	Descripción	Tiempo (Minutos)
Prueba 06	Prueba de funcionamiento del procedimiento manual	27.12
Prueba 07	Prueba de funcionamiento del procedimiento con el sistema	2.27

Tabla 5.4.4.3 Comparación resultados almacén

5.4.5 Modulo Compras

- **Prueba 08.-** Prueba de generación de órdenes de compra con el procedimiento manual

N°	Descripción	Tiempo
1	Listar insumos que se encuentran por debajo de la cantidad mínima	30 min
2	Generar Orden de compra	10 min
3	Buscar orden de compra por fecha	5 min
4	Buscar orden de compra por proveedor	5 min
	Tiempo total	50 min

Tabla 5.4.5.1 Prueba compras manual

- **Prueba 09.-** Prueba de generación de órdenes de compra utilizando el sistema:

N°	Descripción	Tiempo
1	Listar insumos que se encuentran por debajo de la cantidad mínima	2 seg
2	Generar Orden de compra	10 seg
3	Buscar orden de compra por fecha	2 seg
4	Buscar orden de compra por proveedor	2 seg
	Tiempo total	0.27 min

Tabla 5.4.5.2 Prueba compras sistema

La comparación de tiempos del sistema y el procedimiento manual.

Prueba	Descripción	Tiempo (Minutos)
Prueba 08	Prueba de funcionamiento del procedimiento manual	50
Prueba 09	Prueba de funcionamiento del procedimiento con el sistema	0.27

Tabla 5.4.5.3 Comparación de resultados compras

5.5 EVALUACION DE RESULTADOS

5.5.1. Contrastación de la hipótesis:

Hipótesis de Investigación e hipótesis nula:

Para poder contrastar la hipótesis de investigación y la hipótesis nula se han obtenido los siguientes resultados de las pruebas realizadas:

Hipótesis de Investigación:

Hi: La gestión de logística de la panadería "Dos estrellas" si mejorará la toma de decisiones con la implementación un sistema de logística aplicando tecnologías de información.

Hipótesis Nula:

Ho: La gestión de logística de la panadería "Dos estrellas" no mejorará la toma de decisiones con la implementación un sistema de logística aplicando tecnologías de información.

Administración de Logística: TIEMPO

Para demostrar la reducción de tiempos medidos se considerando los procesos utilizando el sistema y con las operaciones manuales.

TPM: Tiempo de proceso de gestión de logística manual.

TPS: Tiempo de proceso de gestión de logística con el sistema.

El porcentaje de reducción del tiempo se calcula de la siguiente forma:

$P\% = 100 - ((TPS * 100) / TPM)$

Si $P\% > 0\%$ se habrá reducido el tiempo de desarrollo

TPM: 72 minutos TPS: 6.37

$P\% = 100 - ((6.37*100)/72)$

$P\% = 100 - (637/72)$

$P\% = 100 - (8.85)$

$P\% = 91.15\ \%$

El porcentaje de reducción de tiempo es de 91.15 % > 0% es decir que SI se ha reducido el tiempo de proceso de gestión de logística en la Panadería dos estrellas.

PASO	DESCRIPCIÓN	PROCESO MANUAL	PROCESO C/SISTEMA
1	Ingreso de Insumos	1min	1min
2	Búsqueda de Insumos	5 min	2 seg
3	Búsqueda de insumos por debajo de la cantidad mínima	30 min	2 seg
4	Elaboración de registro de Salida	5 min	1 min
5	Elaboración de orden de compra	10 min	10 seg
6	Elaboración de un pedido	2 min	1.5 min
7	Búsqueda de pedidos	5 min	2 seg
8	Ingreso de clientes	2 min	1 min
9	Búsqueda de clientes	5 min	3 seg
10	Ingreso de Proveedores	2 min	1.5 min
11	Búsqueda de proveedores	5 min	3 seg
	Tiempo total (minutos):	**72**	**6.37**

Tabla 5.4.1.1. Tiempos

Calidad del Proceso de logística: NIVEL DE SATISFACCIÓN CON EL SISTEMA

En cuanto al nivel de satisfacción con el sistema se realizó dos tipos de encuestas, aplicadas a los usuarios directos del sistema (Administrador y Vendedor) y a los clientes como usuarios secundarios respectivamente.

Según los resultados de las encuestas (presentadas detalladamente en anexos)

NS: Nivel de satisfacción Ri: Respuesta i

Se mide el nivel de satisfacción final de la siguiente manera:

NSi: $\sum$ (Ri)/i donde NS= $\sum$ Ni/i

Administrador: NS1= (5+5+4+5+5+5)/6

NS1 = 4.83

Vendedor:NS2 = (5+4+5+4+5+4)=6
NS2= 4.5

Cliente 1:NS3 = (4+4)/2
NS3 = 4

Cliente 2:NS4 = (5+5)/2
NS4=5

Donde NSF será:
NSF= (4.83+4.5+4+5)/4
NSF= 4.58

En una escala del 1 al 5. Siendo un nivel satisfactorio de los usuarios (tanto primarios como secundarios) favorable para la investigación.

El desarrollo de las encuestas se podrá visualizar en el punto de Anexos del presente informe.

5.5.2. Análisis de Resultados:

La presente investigación está relacionada a mejorar los procesos de gestión de logística en la panadería DOS ESTRELLAS de la ciudad de Piura, por ello el uso del sistema informático es primordial en toda organización, al iniciar este proceso de investigación se realizó una serie de evaluaciones al proceso de logística y a los actores involucrados de la investigación, como son el administrador, almacenero y vendedor.

Los resultados obtenidos son favorables a la investigación pues se está optimizando en un 91.15% en tiempos del proceso de gestión de logística. En cuestión con el nivel de satisfacción el 100% de los encuestados manifestaron que el implementar un sistema Informático de gestión de logística en la panadería DOS ESTRELLAS, mejorará sus operaciones, mientras que el 0% manifiestan que no los mejora. El desarrollo de las encuestas que se aplicó nos permitió visualizar de manera precisa la importancia del uso de un sistema informático.

Teniendo como resultado final que la implementación del sistema informático el administrador de la panadería DOS ESTRELLAS ayuda al proceso de logística de la empresa y por ende a la toma de decisiones de los administrativos de la empresa.

Es por este motivo que afirmamos que el uso del sistema informático mejorara el proceso de gestión de logística en la panadería DOS ESTRELLAS dando por verdadera la hipótesis planteada al inicio de la investigación.

CONCLUSIONES

- Al culminar el proyecto sobre el diseño e implementación de un sistema informático para mejorar el proceso de gestión de logística en la panadería DOS ESTRELLAS se puede afirmar que los objetivos planteados al inicio del desarrollo del proyecto fueron cumplidos de manera satisfactoria.
- La implementación del sistema informático para mejorar el proceso de gestión de logística en la panadería DOS ESTRELLAS permitió solucionar los problemas de almacenamiento, recuperación y búsqueda de información correspondiente productos e insumos de la panadería en mención.
- El diseño modular que tiene el sistema facilita la administración entendimiento del mismo haciendo más fácil la integración de otros módulos o componentes para su crecimiento.
- Como en toda empresa se hace necesario seguir los estándares de desarrollo de sistemas los cuales ayudan a llevar de manera más organizada la información; poder especificar los contenidos que se necesitan visualizar en el sistema y lograr que los beneficiarios se acoplen sin mayor dificultad en su manejo.
- El uso de la metodología de desarrollo RUP, conjuntamente con el lenguaje UML y el manejo de los conceptos de la programación orientadas a objetos, propiciaron que el desarrollo del sistema sea entendible, sostenible e incremental.

RECOMENDACIONES

- Se recomienda tener en cuenta el uso del software como alternativa de desarrollo del sistema, para así beneficiamos de sus ventajas en cuanto a conceptos de independencia, costo y facilidad de desarrollo e implementación, puesto que las herramientas que provee el software libre están muy maduras y capaz de satisfacer las necesidades del desarrollador.
- Para el sistema crezca hasta un nivel gerencial y estratégico, deberán tener en cuenta en proyectos de desarrollos de módulos de gestión, que estos emitan reportes que sea capaz de hacer ver cómo va el giro del negocio, tenencias y además ayude a tomar decisiones a nivel estratégico.
- Los requerimientos de hardware que se pide, según la sección técnica de análisis de factibilidad y el diagrama de despliegue, son mínimos; pero se recomienda que mientras más capacidad tenga el servidor mejor performance tendrá el funcionamiento del sistema.
- Realizar una continua actualización de información y preparación en el manejo del Sistema, por parte de los usuarios pertenecientes a la Empresa.
- Se sugiere a largo plazo la implementación de un sistema web, que permita enlazar la panadería principal y sus sucursales, según el crecimiento del negocio con la finalidad de brindar un mejor servicio y administrar los recursos en conjunto.

BIBLIOGRAFIA

- Balarezo S, Yana M, Ramos Y. (2013). Modelado del Negocio: Parte 1.Modelos de casos de uso del negocio. Versión Electrónica. Recuperado de http://booksproyect.googlecode.com/files/S02-1%20Modelado%20del%20negocio%20(Modelo%20de%20Casos%20d e%20Uso%20del%20Negocio).ppt

- Coate Rosales Edmundo, Saavedra Medina Nidia Carolina (2010). "UML_y_RUP", Versión electrónica. Recuperado de https://www.google.com.pe/#q=para+que+se+utilizan+los+diagramas+de+casos+de+uso+del+rup.

- Fowler Martin; Scott Kendall (1999). "UML GOTA A GOTA". Versión electrónica. Recuperado de http://books.google.com.pe/books?id=AL0YkFeaHwIC&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

- Levano Rodriguez D. (2013), "DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS"

- Martinez Rafael. (2013). Sobre PostgreSQL. 2015, de PostgreSQL Global Development Group Sitio web: http://www.postgresql.org.es/sobre_postgresql

- Programa de Formación empresarial para el comercio de minorista especializado en la comunidad de Madrid (2011). "COMERCIO DE PANADERÍA Y PASTELERÍA". España: Ediciones MadridInnova

- Sanchez Jorge (2004). Java2 - Swing, Threads, programación en red, JavaBeans, JDBC y JSP / Servlets.

- Wrong Henry. (2010). Encriptación de Datos MD5. 2015, de UPAO Sitio web: https://sites.google.com/a/upao.edu.pe/hwongu/mundo-java/tips-de-codificacion/encriptacion-de-datos-md5

ANEXOS

MANUAL DE USUARIO SISTEMA DE LOGISTICA DE PANADERIA "DOS ESTRELLAS"

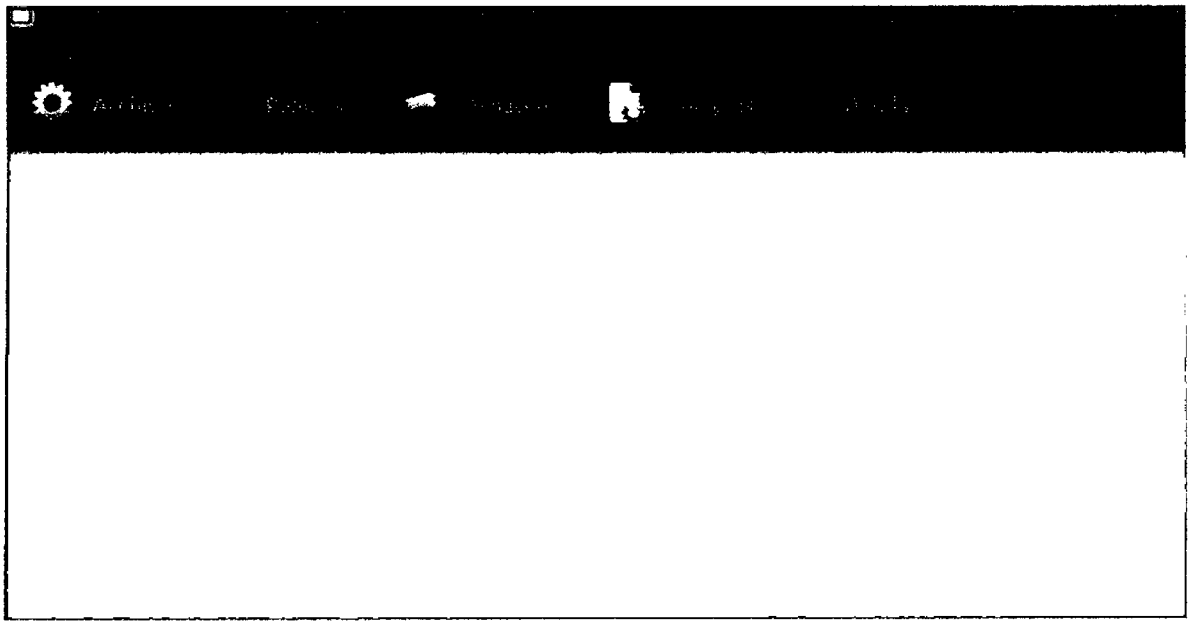
1.- INGRESO AL SISTEMA

Como acceder al Sistema de logística de la Panadería "DOS ESTRELLAS"

1. Ubíquese en el ícono de acceso directo del Sistema y haga doble clic.

2. A continuación aparecerá una ventana que permitirá acceder al sistema mediante un usuario previamente creado que en los próximos capítulos se detallará como hacerlo.

3. Posteriormente aparecerá la pantalla de todos los Servicios que ofrece este sistema.

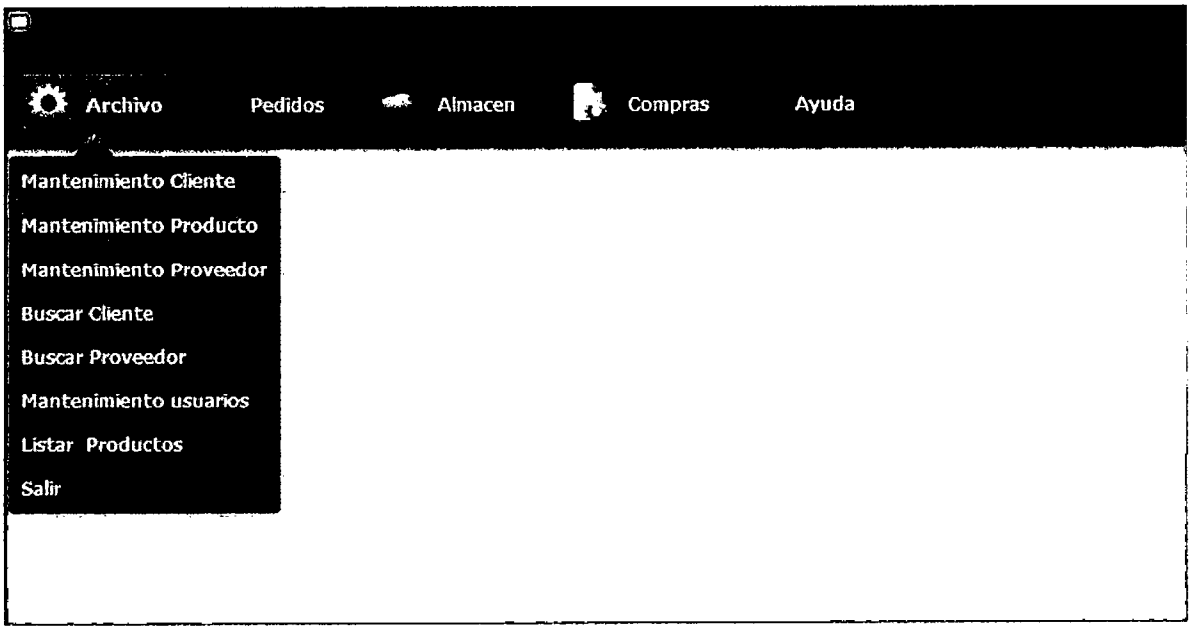


Seguidamente explicaremos las opciones que posee nuestro sistema:

4. Modulos:

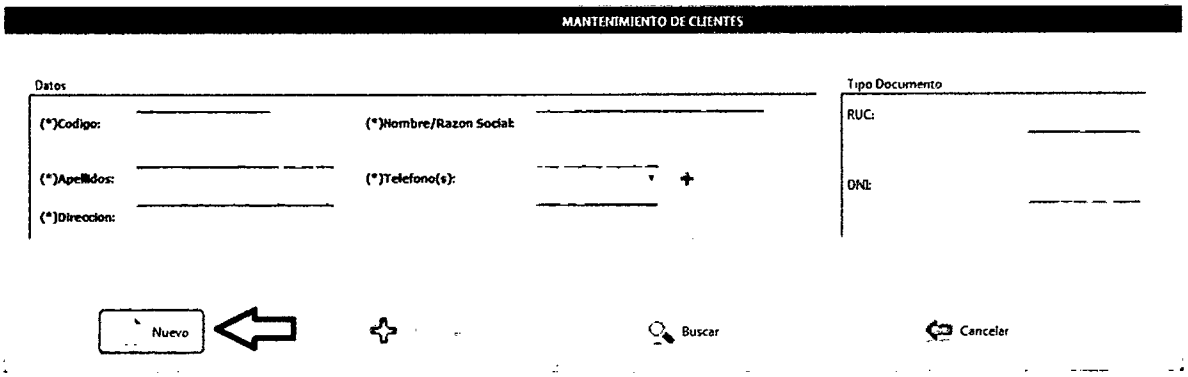
I. MODULO ARCHIVOS

En el módulo de archivos encontraremos las ventanas de mantenimiento de cliente, proveedores y productos finales de la panadería.

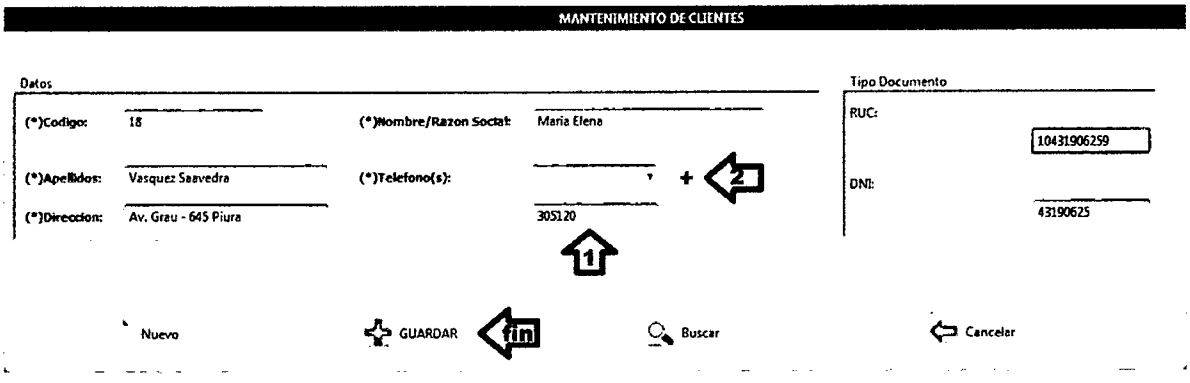


I.1.- Mantenimiento de cliente: En esta ventana se podrá hacer el ingreso de los clientes,búsqueda y actualizaciones.

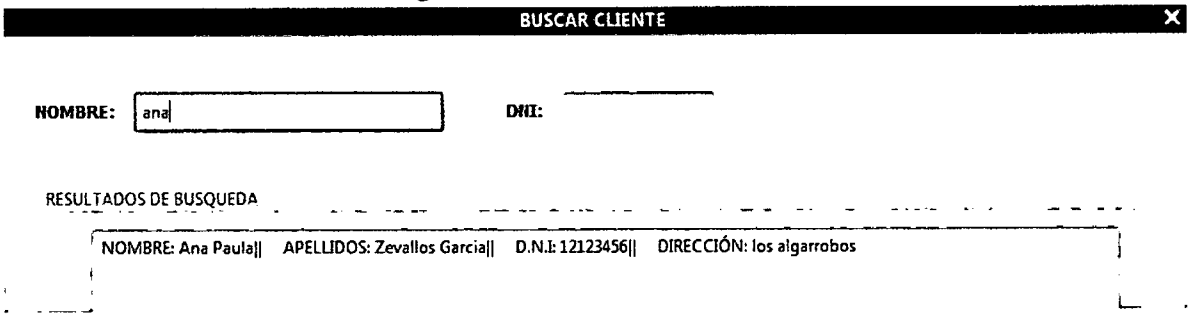
Para ingresar un cliente, hacer clic en NUEVO:



Seguidamente completar los datos que se piden en el formulario,

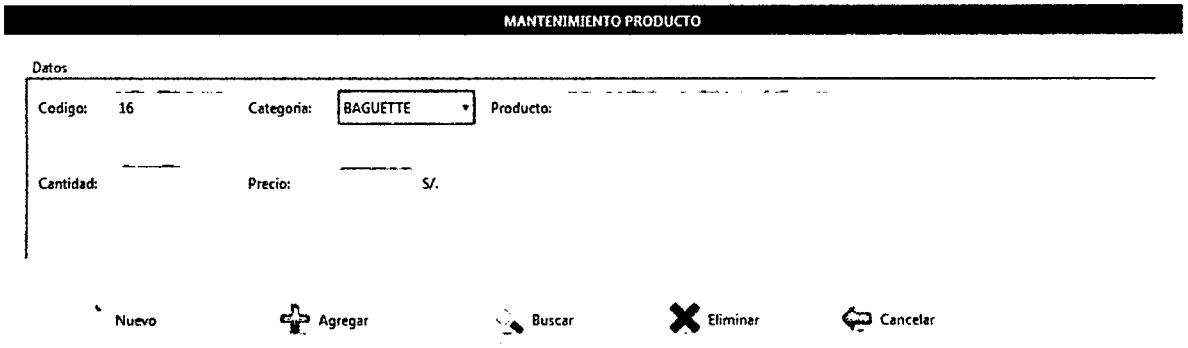


Al dar clic en buscar saldrá la siguiente ventana



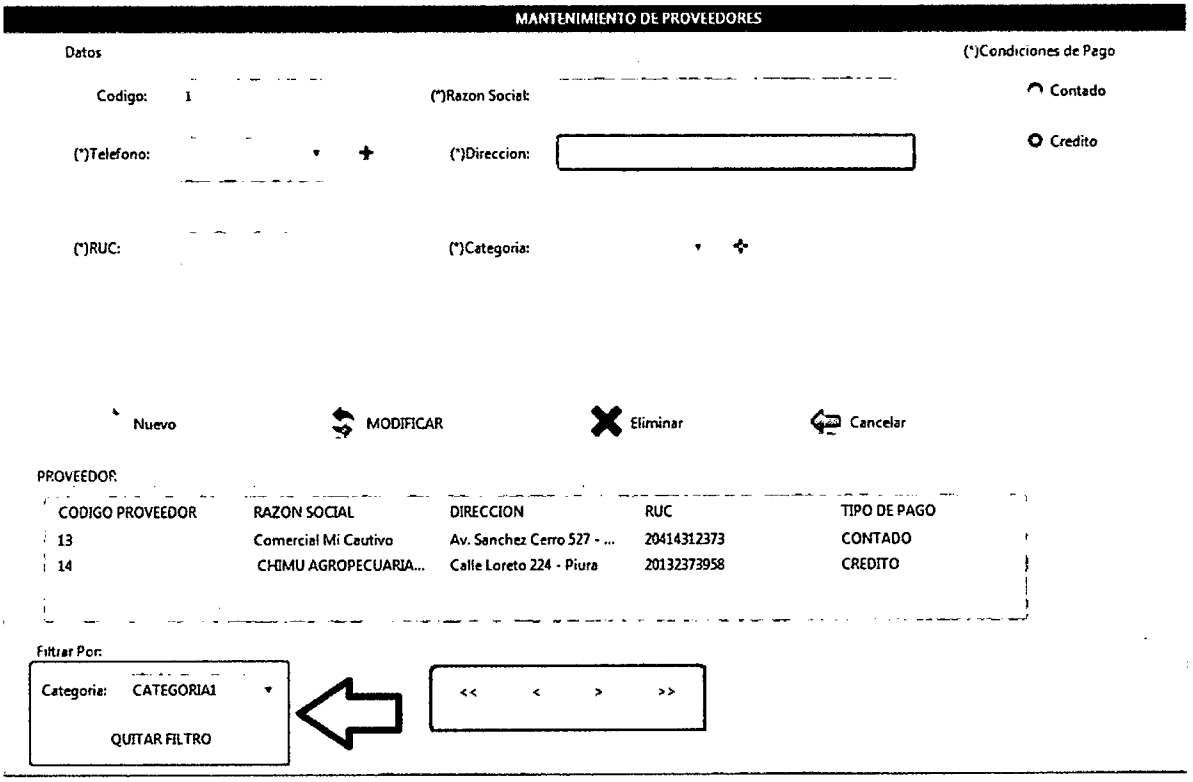
Dar doble clic si se desea visualizar los datos o modificar, se cargarán los datos del cliente elegido. Si se desea modificar dar clic en GUARDAR.

I.2.- Producto: Tiene las mismas funciones de Mantenimiento de cliente con algunos datos diferentes para ingresar.

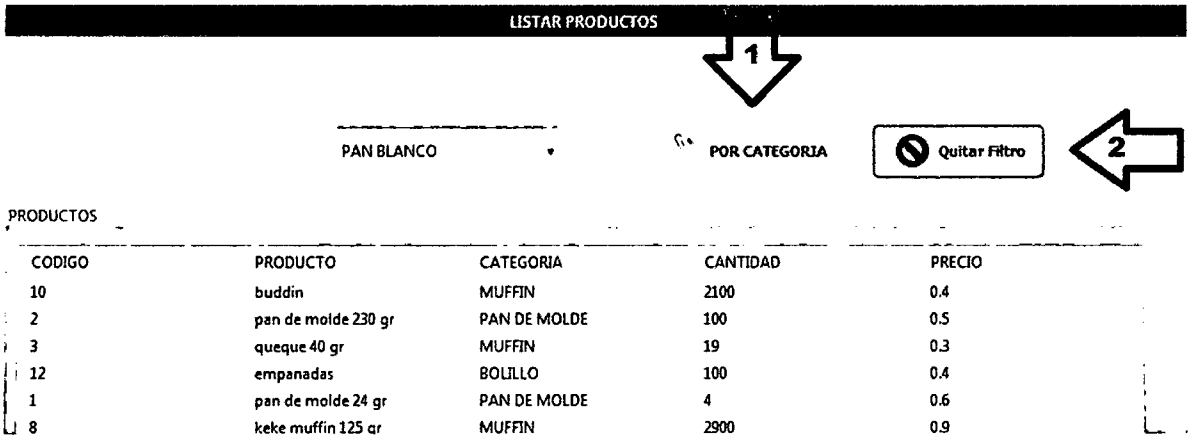


I.3.- Proveedor: para el registro de proveedores, al igual que las anteriores ventanas se tendrá una ventana en la que nos proporcione todos los campos a completar.

A diferencia de la búsqueda de las ventanas anteriores, será que en esta ventana también podremos buscar a los proveedores por **categorías** para una mejor administración de datos.

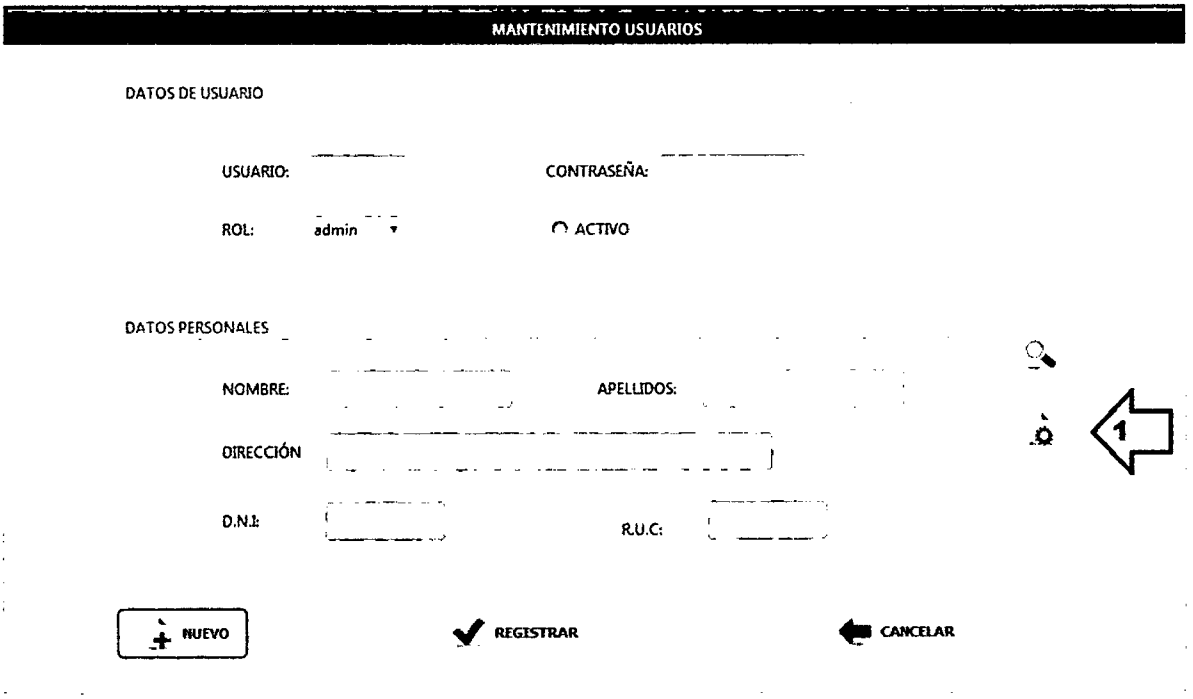


I.4.- Listado de Productos.- Para Listar los productos finales de la panadería se ingresará al listado de Productos mediante el cual se podrán listar por categorías, después de elegir la categoría hacer clic en el botón CATEGORÍA (1) y se filtrará según la opción elegida. Si se desea volver a visualizar el total de productos sólo se presionará el botón QUITAR FILTROS (2) para ejecutar.

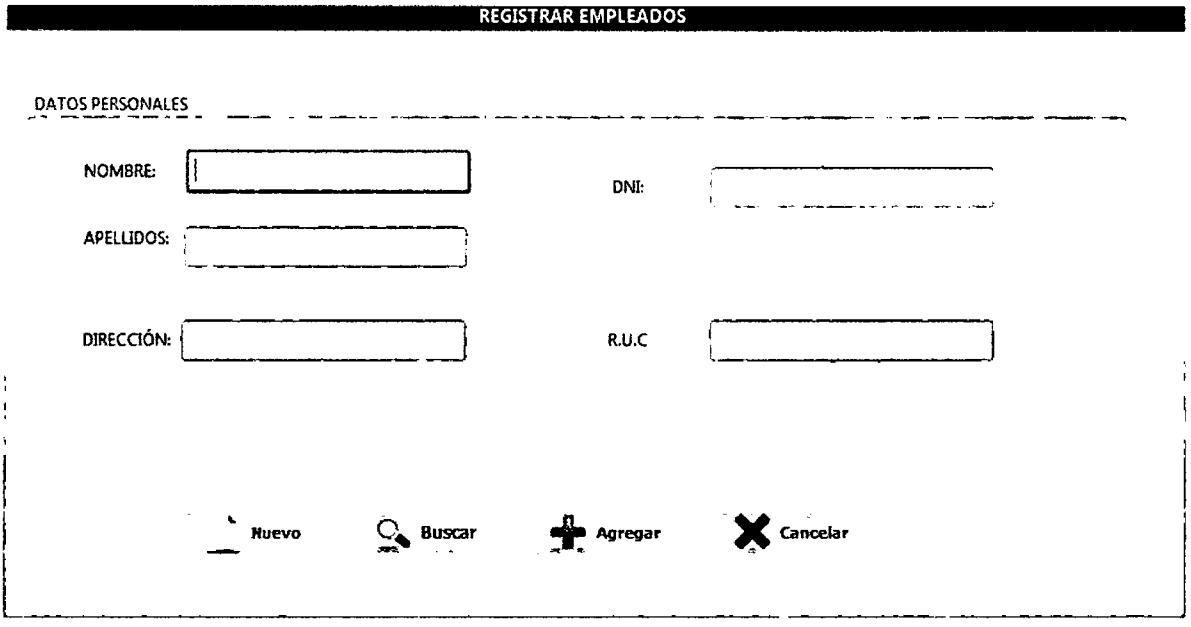


I.5.- Mantenimiento de Usuarios:

Para ingresar un usuario, si el empleado no se encuentra ingresado dar clic en el boton INGRESAR EMPLEADO (1)

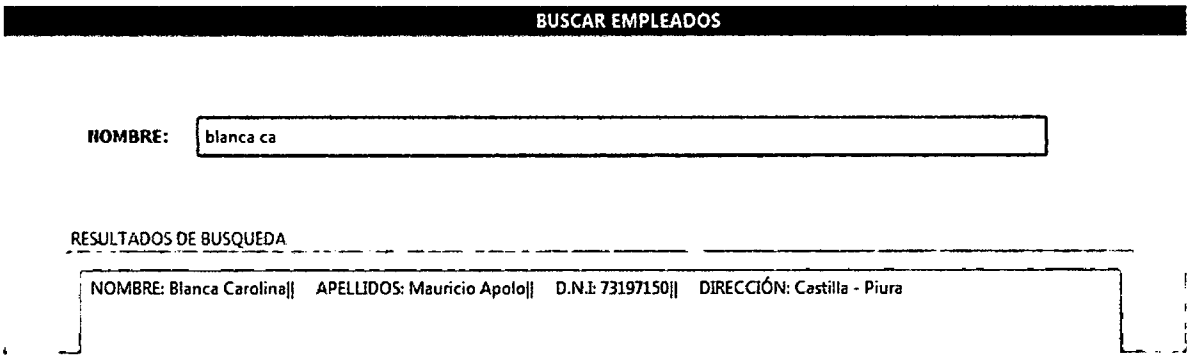


Ingresar los datos en la nueva ventana y dar clic en AGREGAR

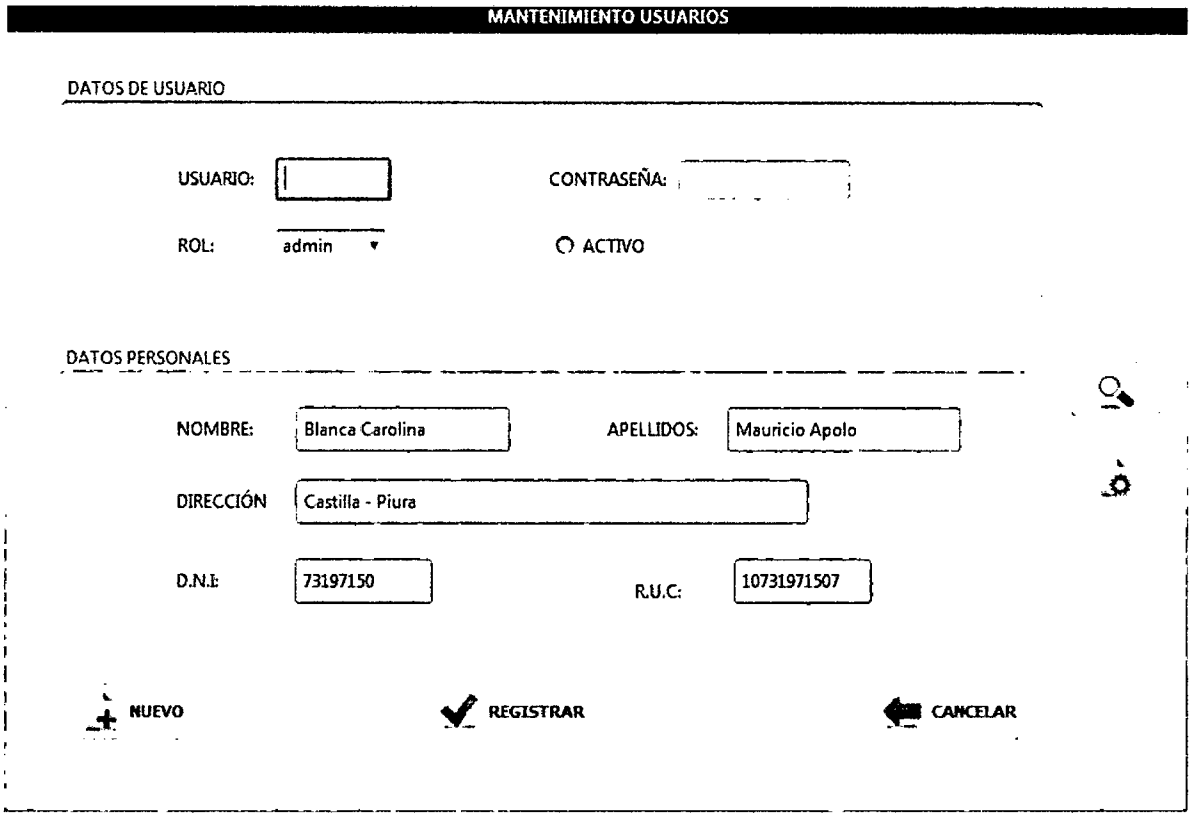


Para cargar los datos y registrar un usuario dar clic en BUSCAR EMPLEADO  de la ventana principal.

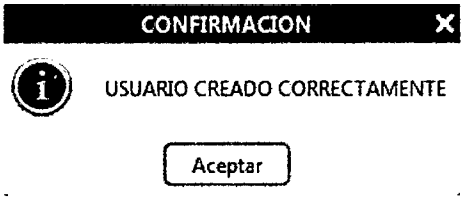
Buscar por nombre el empleado al cual se necesita crear el usuario seleccionar dando ENTER:



Automáticamente se cargan los datos en la ventana principal y proceder a digitar los datos restantes.



Finalizar presionando el botón registrar.

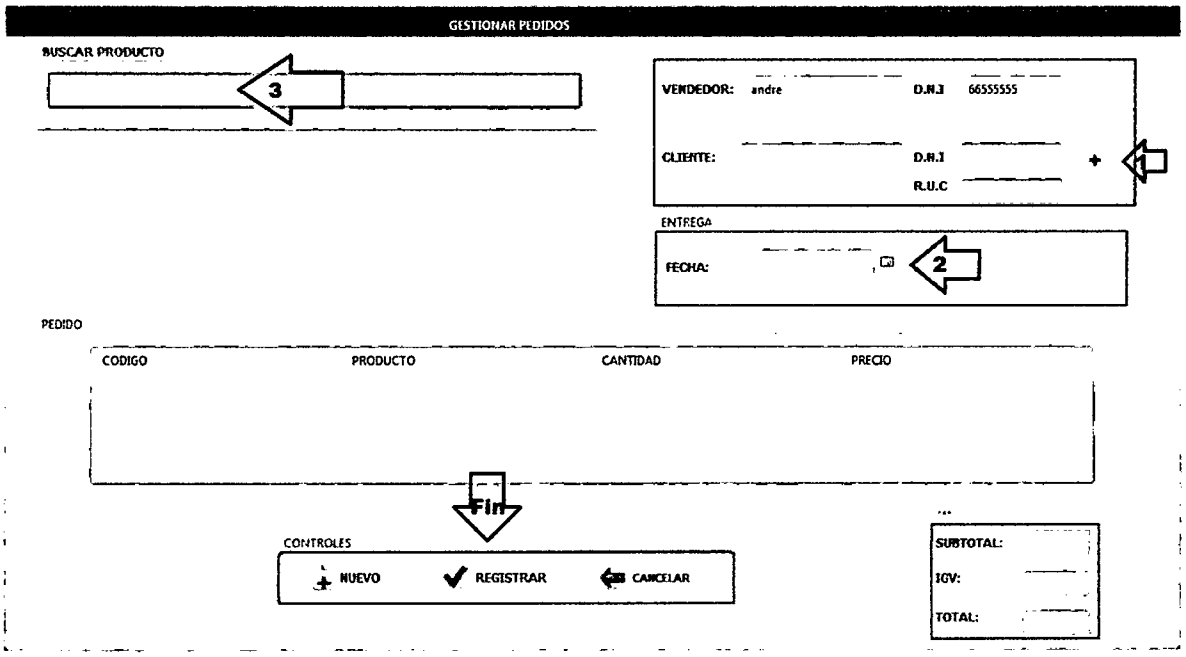


II.- MODULO DE PEDIDOS

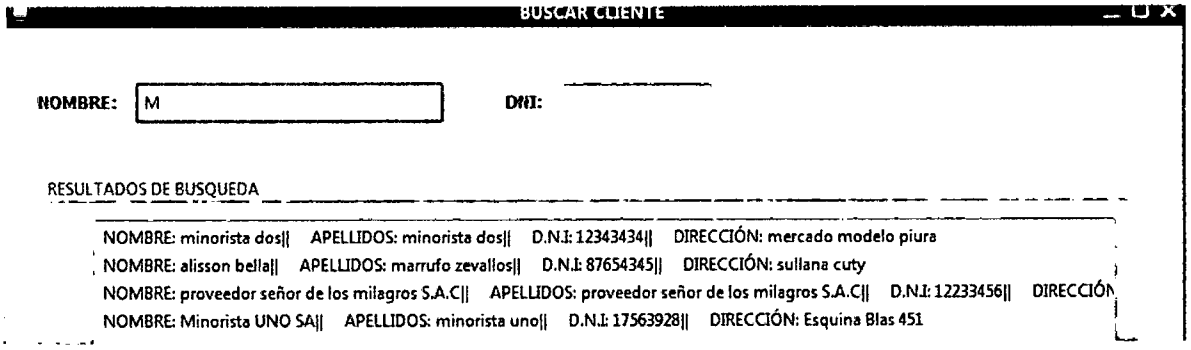
II.1.- Gestionar Pedidos.- En el siguiente módulo se registrará todos los pedidos a atender.

Para empezar a registrar un pedido dar clic en el botón NUEVO.

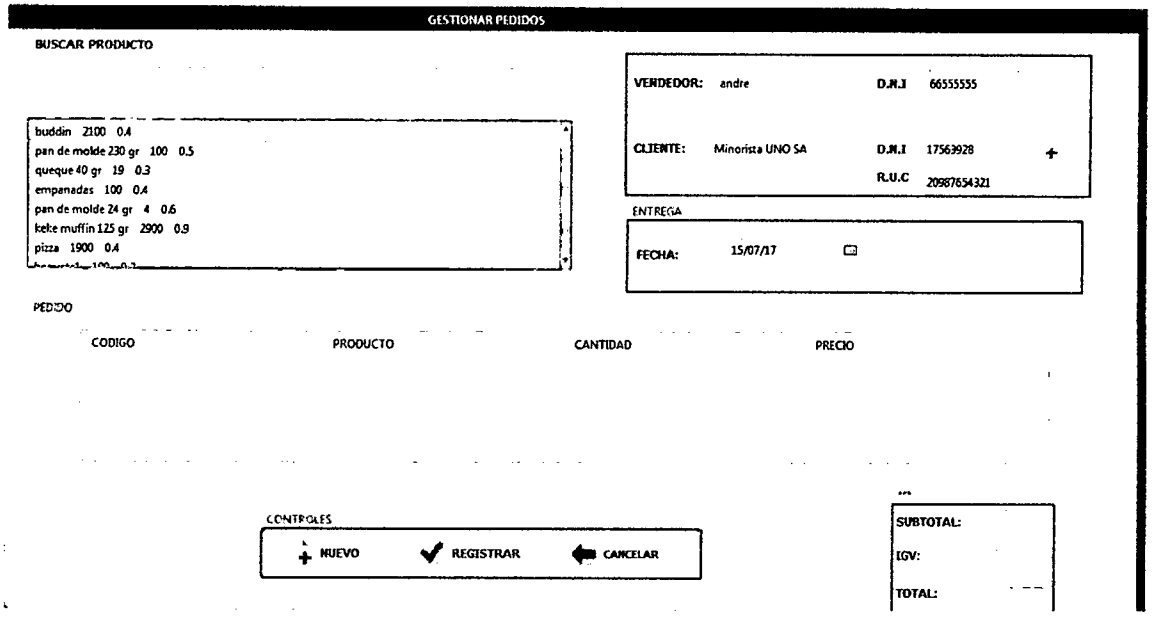
A continuación ingresar los datos tal y como se detalla el orden de las flechas.



En el paso 1 que es el cliente, dar clic al botón  y automáticamente aparecerá la siguiente ventana.

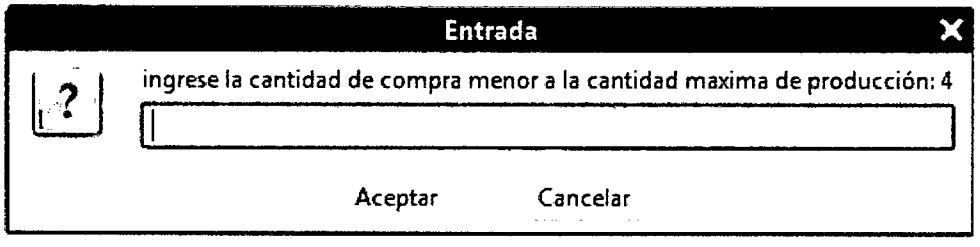


Digitar el nombre y/o DNI del cliente al que se desea registrar el pedido seleccionarlo y dar ENTER. Automáticamente los datos se cargan en la ventana anterior.

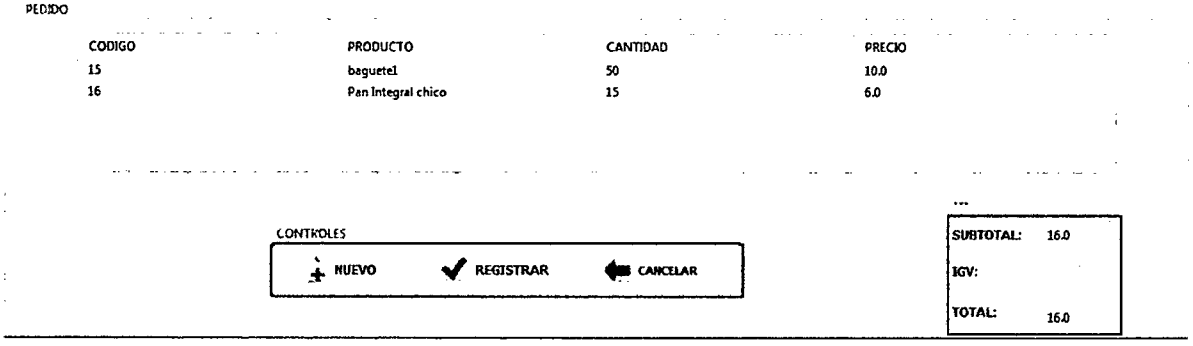


Para ingresar la fecha del mismo modo de cliente hacer clic en el botón y elegir la fecha requerida.

A continuación para ingresar los productos del pedidos digitar el nombre de lo que se desea ingresar, presionar ENTER y aparecerá la ventana:

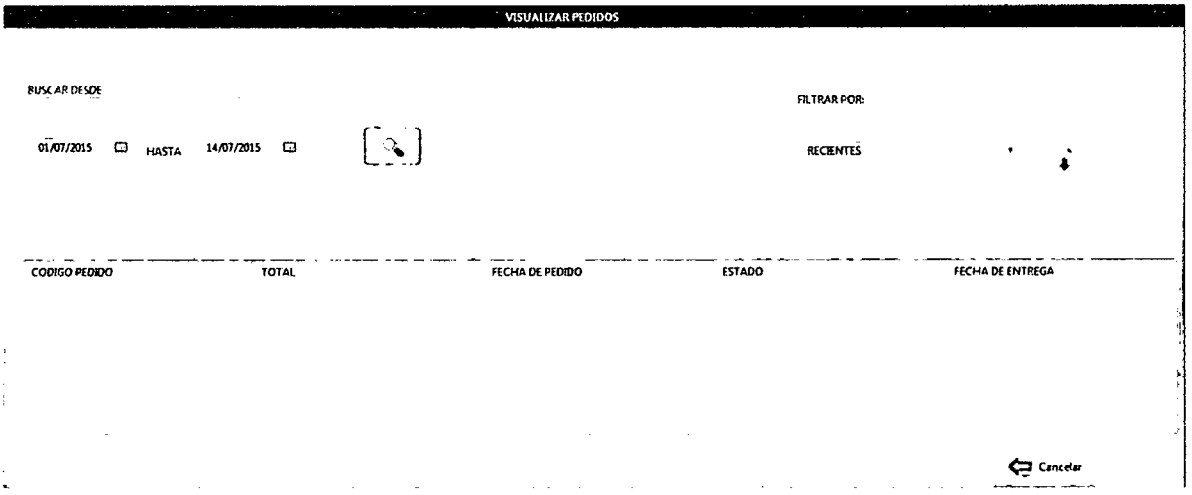


En la cual ingresar la cantidad necesitada. Los productos se cargarán en la tabla que se presenta en la parte inferior de la ventana.

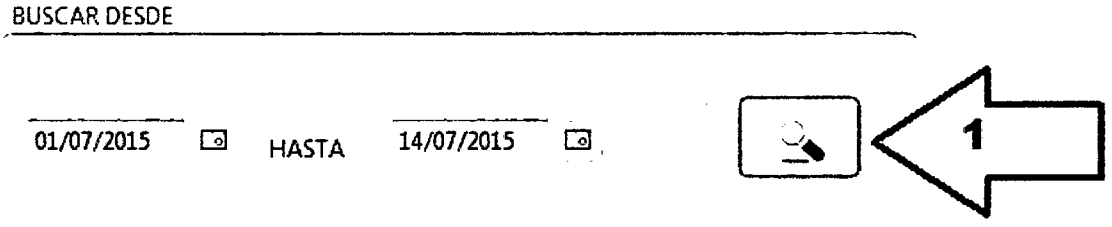


Como paso final presionar el botón REGISTRAR para guardar el pedido.

II.2 Visualizar pedidos.- Esta ventana permite la búsqueda de Pedidos según fecha de atención, de ingreso o condición.

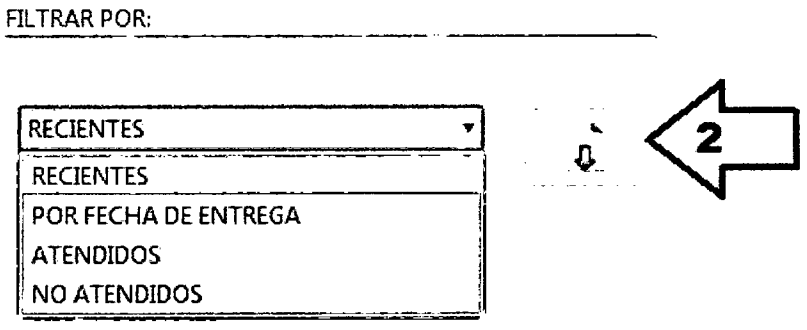


Después de seleccionar las fechas dar clic en el botón BUSCAR para que los pedidos aparezcan en la tabla de la parte inferior de la ventana según la fecha de atención.

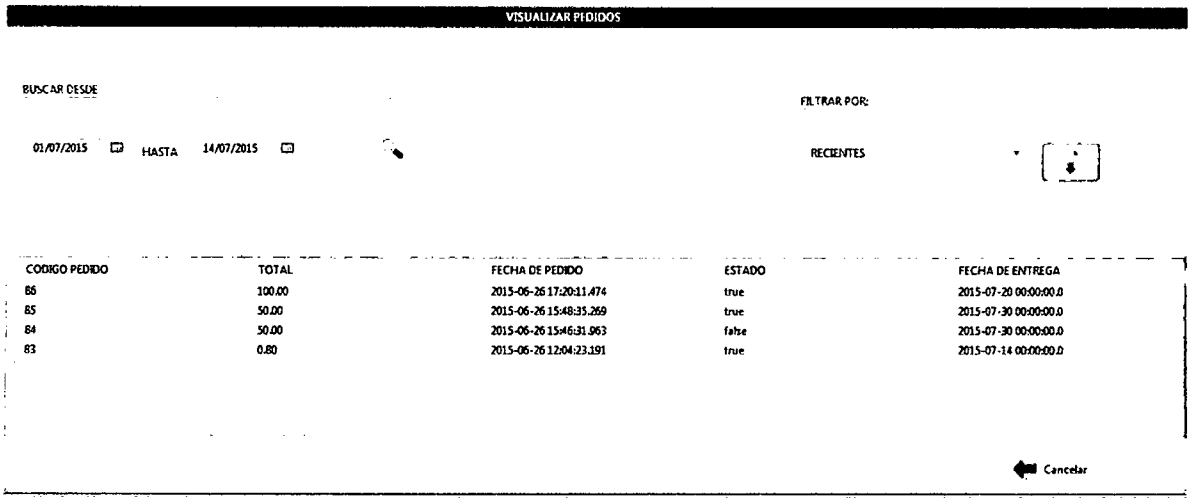


En la parte derecha de la ventana aparecen los filtros de búsqueda según fecha de ingreso, y por estado de pedido (atendido o no atendido).

Seleccionar el tipo de filtro que se desea aplicar y dar clic en botón BUSCAR (2).

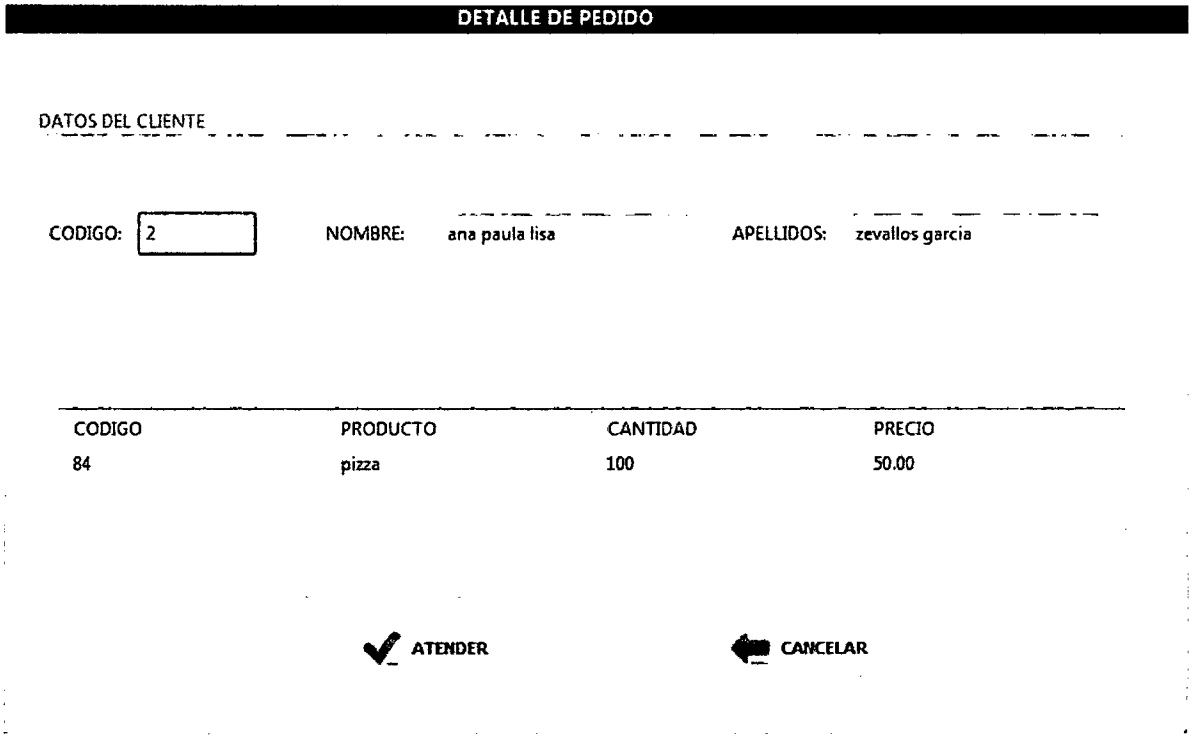


Los datos generales de los pedidos se cargarán en la tabla de la siguiente manera:



Para visualizar le contenido de cada uno de los pedidos sólo elegir el pedido deseado y dar ENTER.

A continuación se mostrará la ventana de detalles:



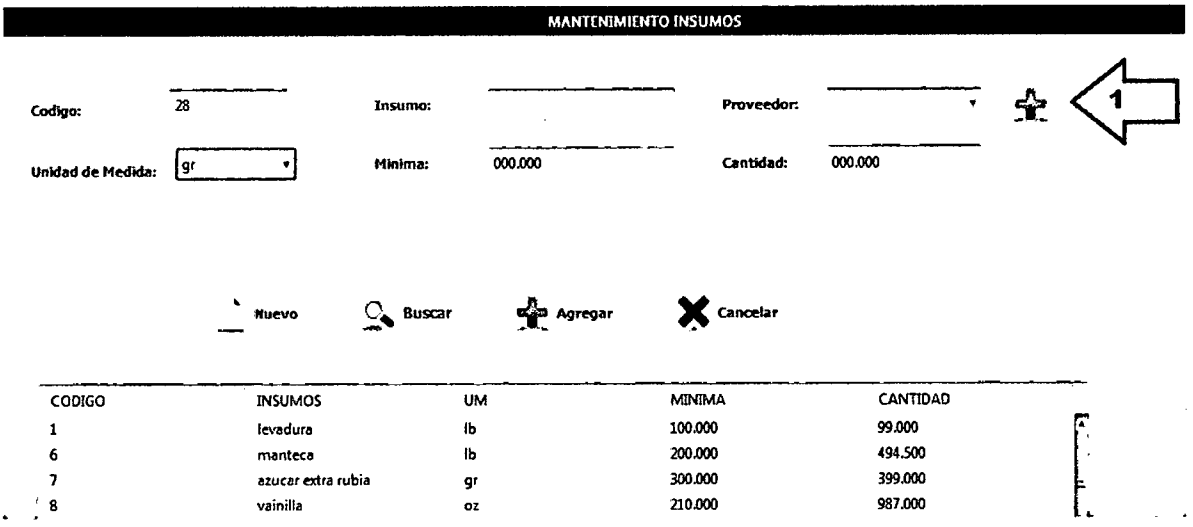
En el caso de pedidos "No atendidos" se atenderá dando clic en el botón 

y este cambiará su estado a ATENDIDO.

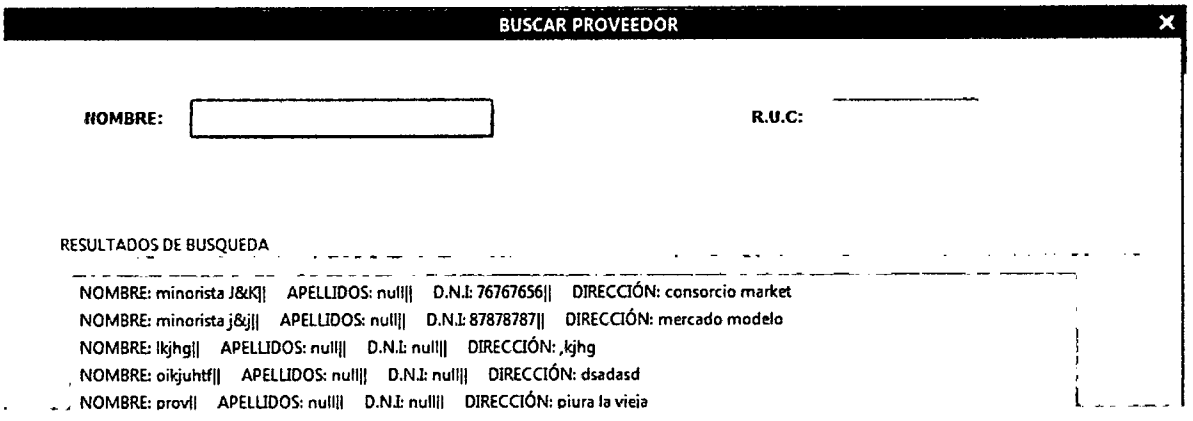
III.- MODULO ALMACEN

III. 1.- Mantenimiento de insumos.- en la presente ventana se registrarán todos los insumos o

materia prima que se encuentra en el almacén:

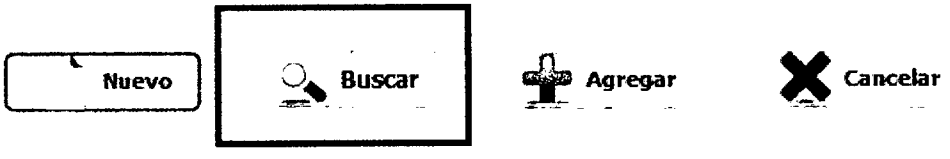


Dar clic en NUEVO para registrar un nuevo insumo. Las demás funciones son muy parecida a las de las anteriores ventanas, para ingresar los proveedores dar clic en el botón  .

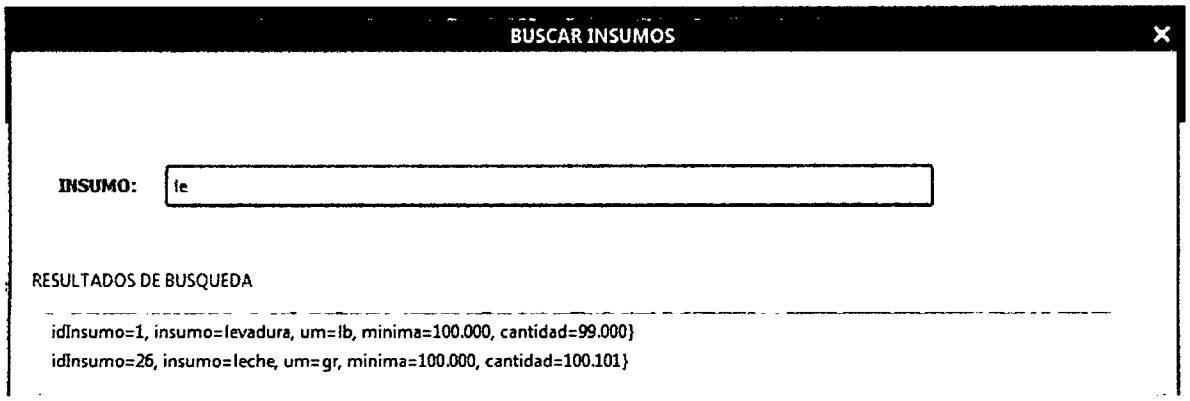


Digitar el nombre del proveedor que abastece del insumo y dar ENTER, si es más de uno, repetir la acción.

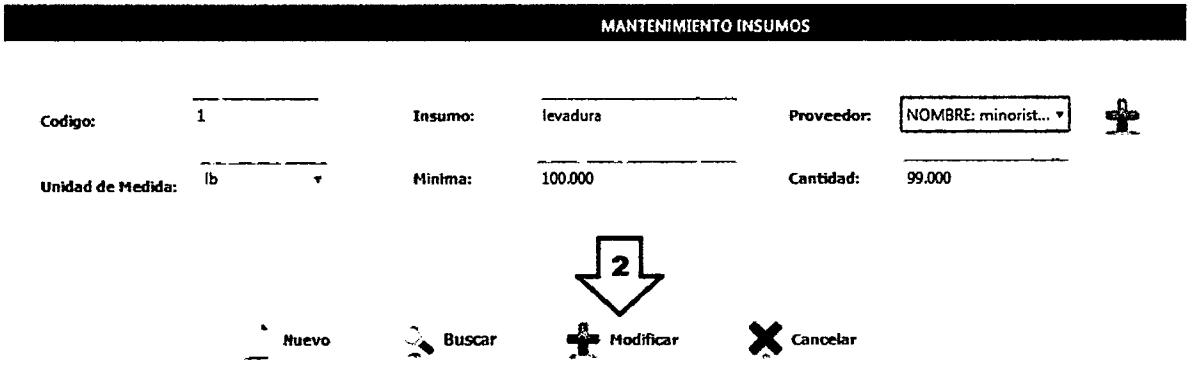
Para la búsqueda seleccionar el botón BUSCAR:



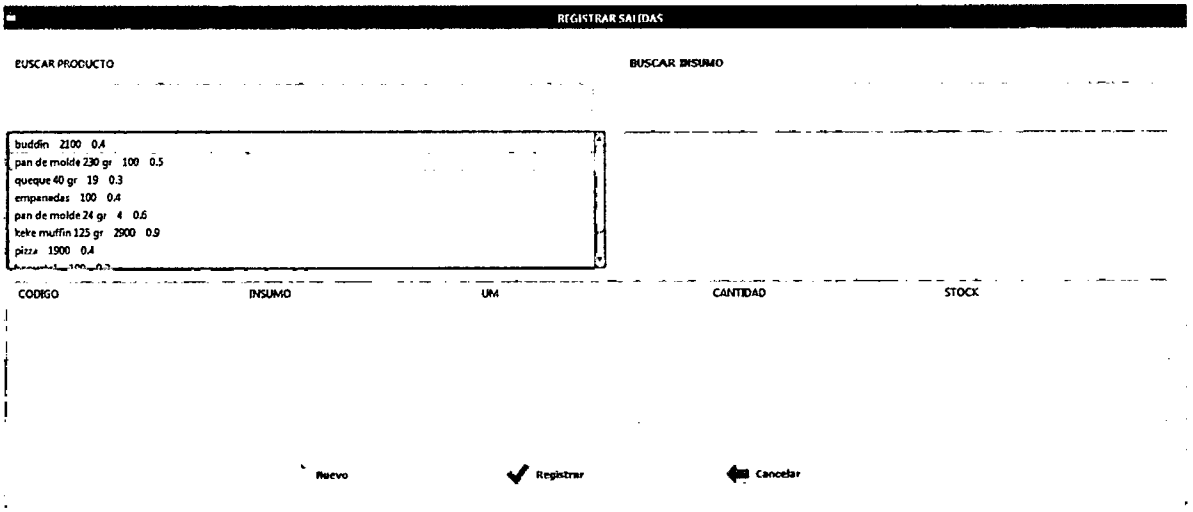
Digitar en la ventana de diálogo que aparece el insumo que desea visualizar y dar ENTER para cargar los datos



Automáticamente se cargan los datos, si se desea modificar algunos de ellos, reescribir la actualización y dar clic en el botón MODIFICAR (2) para guardar los nuevos datos.



III.2. Registrar Salidas. Para cada jornada de producción se registrarán las salidas de insumos del almacén según la producción del día 2



Digitar el producto a producir y dar enter, seguidamente aparecerá la siguiente ventana:

DATOS PRODUCTO

CODIGO: 2 PRODUCTO: pan de molde 230 gr

INGREDIENTES

CODIGO INSUMO	INSUMO	UM	CANTIDAD NECESARIA
8	vainilla	oz	4.000
24	azucar blanca	gr	100.000
26	leche	gr	0.300

Aceptar cancelar

A continuación se visualiza la lista de ingredientes y cantidades necesarias para su preparación.
Dar clic en botón ACEPTAR.
Ingresar la cantidad que se desea producir y dar clic en ACEPTAR

Entrada

INGRESE LA CANTIDAD DEL PRODUCTO PAN DE MOLDE 230 GR A COTIZAR

1000

Aceptar Cancelar

Si se desea agregar más insumos adicionales, digitar en la parte superior derecha los insumos:

REGISTRAR SALIDAS

BUSCAR PRODUCTO

queque 40 gr 19 0.3
empanadas 100 0.4
pan de molde 24 gr 4 0.6
keke muffin 125 gr 2900 0.9
pizza 1900 0.4
baguetel 100 0.2
Pan Integral chico 20 0.4

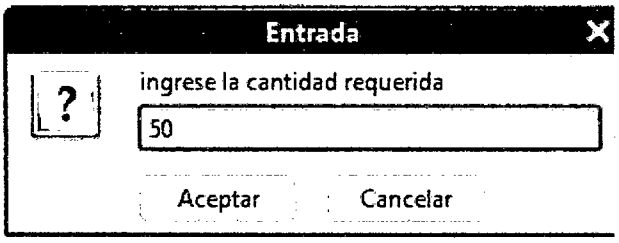
BUSCAR INSUMO

fru

idInsumo=23, insumo=frutilla, um=gr, minima=222.222, cantidad=333.444)

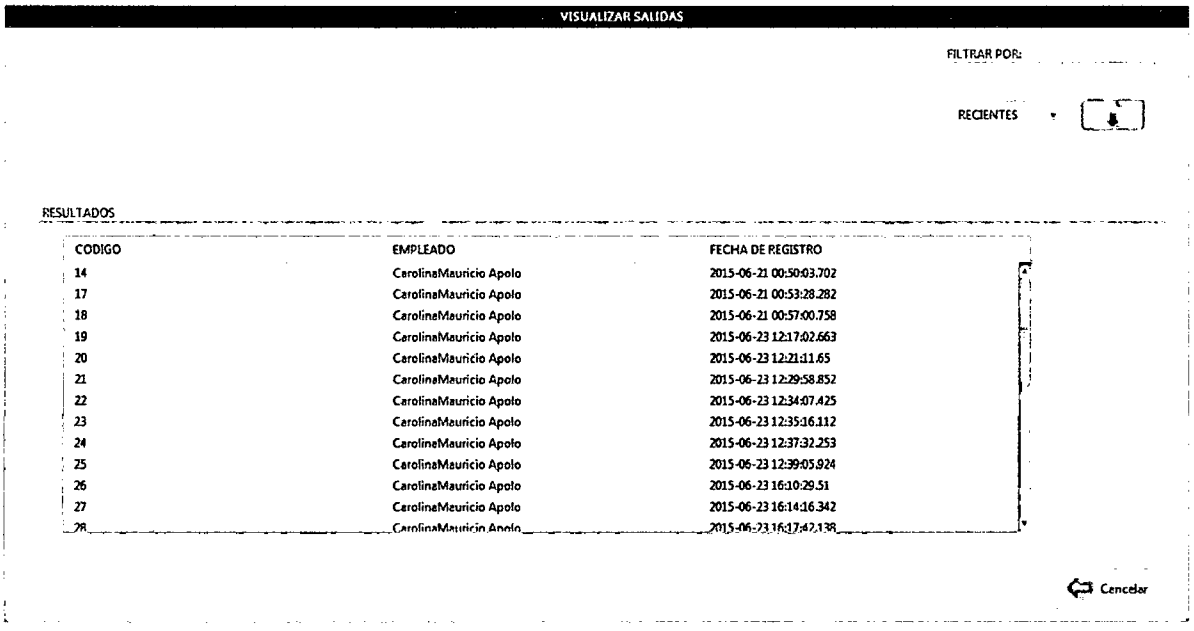
CODIGO	INSUMO	UM	CANTIDAD	STOCK
8	vainilla	oz	400.0000	587.0000
26	leche	gr	30.0000	70.1010
6	manteca	lb	200.0	294.500

Dar enter e ingresar la cantidad deseada para ser cargada en la salida:

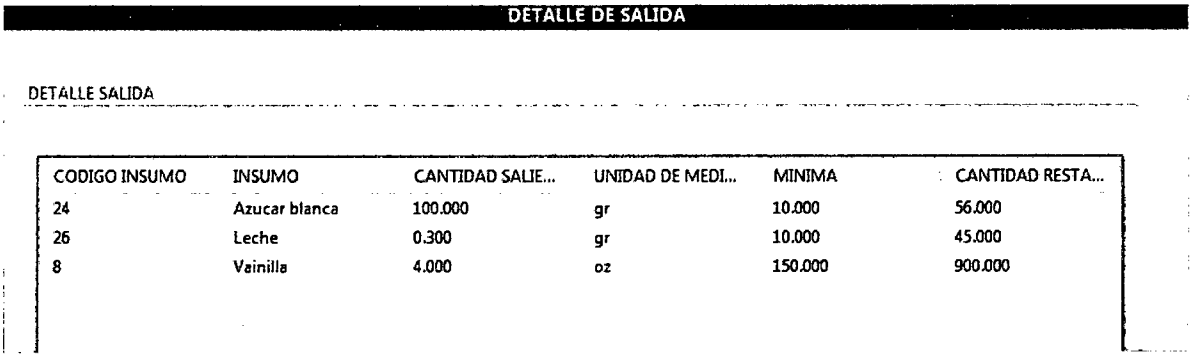


Al finalizar dar clic en el botón REGISTRAR y quedará guardado el registro.

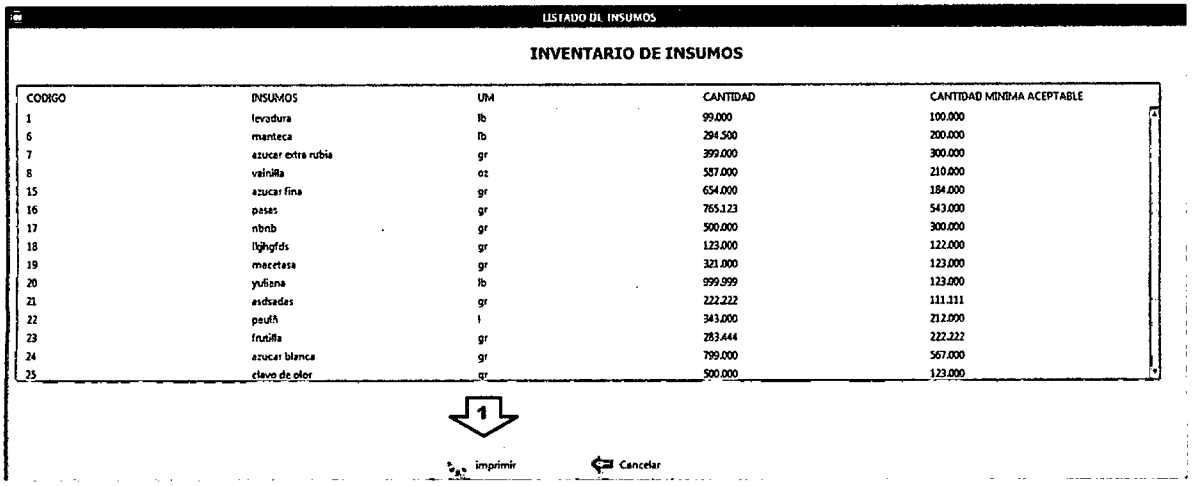
III.3.- Visualizar Salidas



Presionar ENTER en la que se desea visualizar el detalle y aparecerá la siguiente ventana:



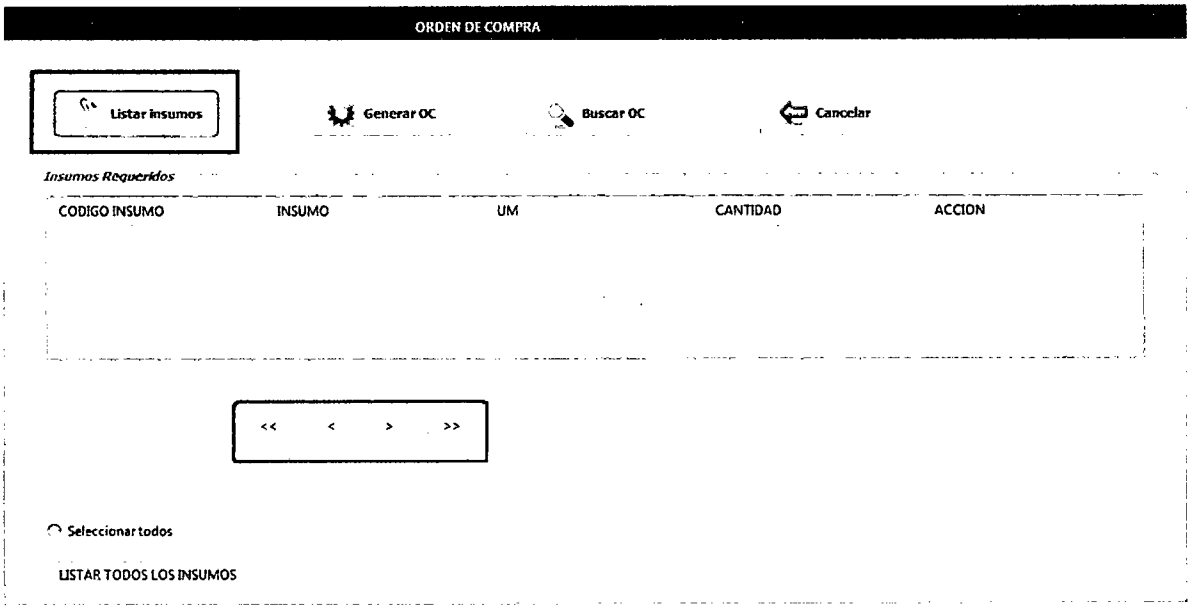
III. 4.- Listado de insumos: Para el inventario se listará la lista total de insumos.



Para sacar el reporte de los insumos existentes en almacén dar clic en IMPRIMIR.

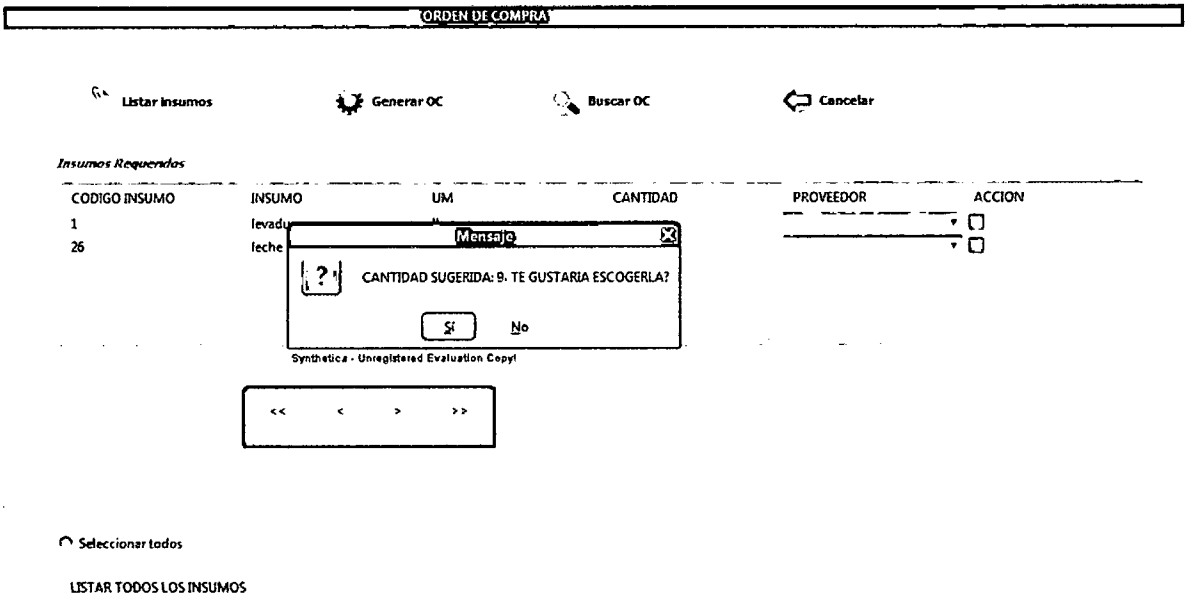
IV.- MODULO COMPRAS

IV. Gestionar Orden de compra.- se mostrará la ventana siguiente



Al dar clic en el botón LISTAR INSUMOS aparecerán en la tabla todos los insumos que se encuentren en su cantidad mínima en espera de que se genere su OC (orden de compra) De la misma manera se pueden listar TODOS los insumos dando clic en el botón que se encuentra en la parte inferior de la pantalla LISTAR TODOS LOS INSUMOS

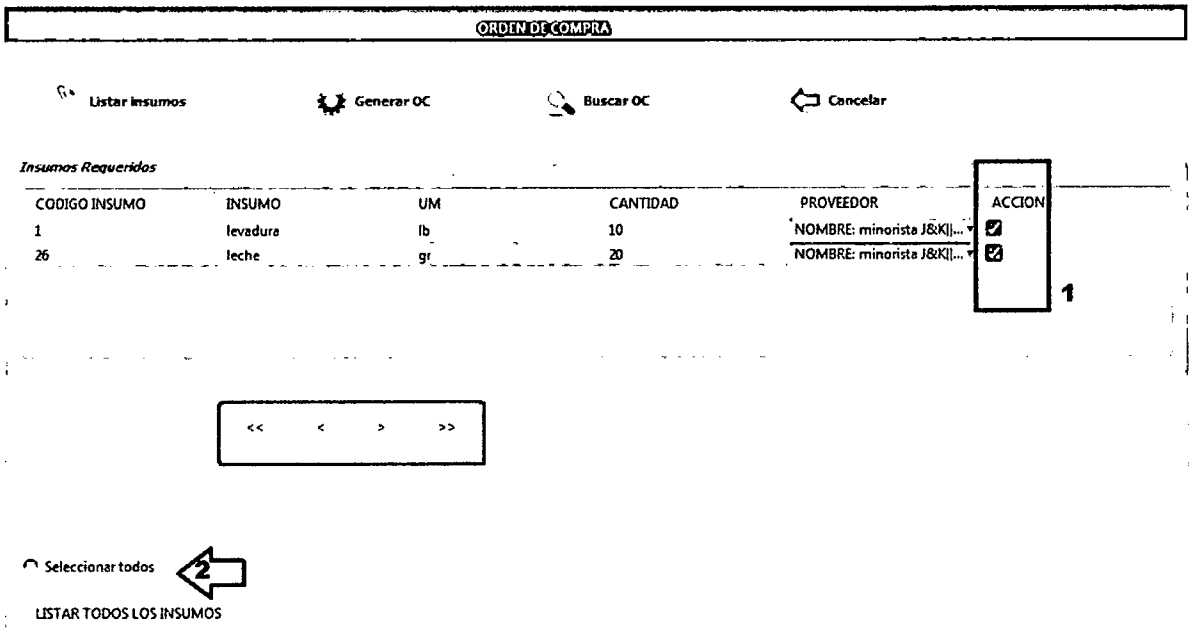
Seleccionar cantidad y aparecerá una cantidad sugerida por el sistema según el historial de compras realizado anteriormente.



Si se acepta la cantidad se cambiará a esta, de lo contrario el usuario puede ingresar la cantidad según requerimiento.

A continuación se agregará el proveedor al cual se le realizará la compra.

Los insumos que será agregados a la OC deberán ser seleccionados dando clic en el cuadrito de la columna ACCION como se muestra en (1):

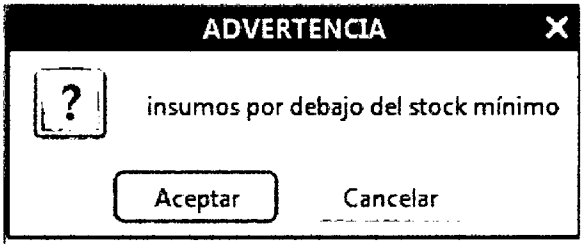


Si los insumos son muchos se tiene la opción SELECCIONAR TODOS (2) en la parte inferior izquierda de la pantalla.

Seguido de eso dar clic en GENERAR ORDEN DE COMPRA. Muestra mensaje de

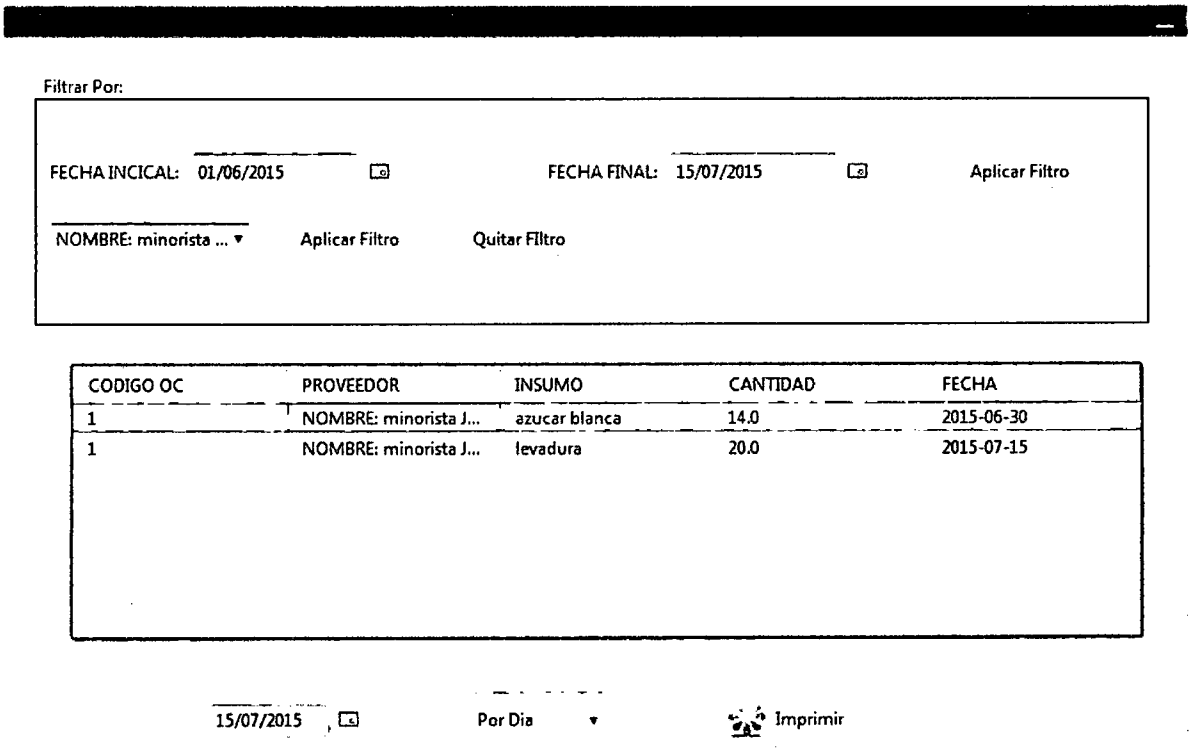
confirmación:

Si hay productos que está por debajo de su cantidad mínima el usuario recibirá una alerta al momento ejecutar el sistema.



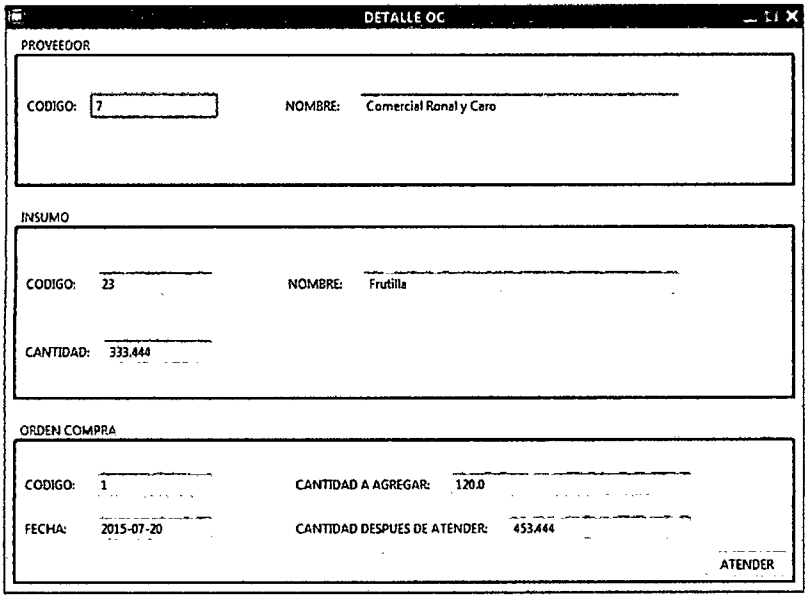
Al aceptar llevará a la ventana para generar la OC, si no se acepta, recordará la alarma cada 30 minutos.

IV.- Visualizar Orden de compra: Esta permite buscar por fecha o proveedor:



Para obtener el reporte de la orden de compra dar clic en IMPRIMIR.

Para atender la OC dar doble clic en el insumo que se desea atender y saldrá la siguiente ventana



Clic en atender y el stock se actualizará.

ENCUESTAS REALIZADAS

Encuesta Usuarios de Sistema de Gestión Logística:

Nombre: _____

DNI: _____

Cargo: _____

A. Pensando en las necesidades que en materia gestión de logística tiene la empresa, ¿Cuán satisfecho está con el sistema desarrollado como herramienta de apoyo? Le pedimos que evalúe en una escala del 1 al 5, donde 1 es muy insatisfecho y 5 es muy satisfecho.
   1. Muy insatisfecho
   2. Insatisfecho
   3. Ni satisfecho, ni insatisfecho
   4. Satisfecho
   5. Muy Satisfecho

B. Luego de haber recibido capacitación sobre el uso del sistema, Ud. diría que...?.
   1. No recibió capacitación
   2. Es difícil Operar el sistema
   3. Tiene un complejidad media
   4. No es Fácil ni difícil
   5. Es fácil operar el sistema

C. En una escala del 1 al 5, donde 1 es muy mala y 5 muy buena, ¿cómo evalúa la velocidad del sistema?
   1. Muy mala
   2. Mala
   3. Ni buena, ni mala
   4. Buena
   5. Muy Buena

D. En una escala del 1 al 5, donde 1 es muy buena y 5 muy mala, ¿cómo evalúa la disponibilidad del sistema (en términos de que no tenga cortes, interrupciones o fallas)?
   1. Muy mala
   2. Mala
   3. Ni buena, ni mala
   4. Buena
   5. Muy Buena

E. En cuanto al funcionamiento general del sistema, Ud. diría que es...?
   1. Muy mala
   2. Mala
   3. Ni buena, ni mala
   4. Buena
   5. Muy Buena

F. Pensando ahora en la capacitación recibida sobre el uso del sistema, Ud. considera que la misma fue... ?
   1. Muy mala
   2. Mala
   3. Ni buena, ni mala
   4. Buena
   5. Muy Buena

_____

Firma

Encuesta Clientes panadería DOS ESTRELLAS:

Nombre: _____

DNI: _____

A. Tipo de cliente:
   Subdistribuidor
   Cliente Final

B. ¿Cuál fue su último pedido realizado en la panadería DOS ESTRELLAS?
   _____

C. En una escala del 1 al 5, donde 1 es muy mala y 5 muy buena ¿Cómo evalúa el avance de la empresa con respecto a efectividad de atención?
   1. Muy mala
   2. Mala
   3. Ni buena, ni mala
   4. Buena
   5. Muy Buena

D. En una escala del 1 al 5, donde 1 es muy puntual y 5 es muy impuntual ¿Cómo se le atendió el último pedido realizado a la panadería.
   1. Muy impuntual
   2. Impuntual
   3. Como siempre
   4. Puntual
   5. Muy puntual

E. Observaciones:

_____

_____

Firma